Towards a More Efficient Use of Computational Budget in Large-Scale Black-Box Optimization

A thesis submitted in fulfilment of the requirements for the degree of
Doctor of Philosophy

Borhan Kazimipour
Master of Science (Artificial Intelligence and Robotics), Shiraz University, Iran
Bachelor of Science (Software Engineering), Shiraz University, Iran

School of Science
College of Science, Engineering, and Health
RMIT University

April 2018

Declaration

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program; any editorial work, paid or unpaid, carried out by a third party is acknowledged; and, ethics procedures and guidelines have been followed.

I acknowledge the support I have received for my research through the provision of an Australian Government Research Training Program Scholarship.

Borhan Kazimipour
School of Computer Science and Information Technology
RMIT University
April 2018

Acknowledgments

I would like to offer my deep gratitude to my knowledgeable, experienced, and genuinely friendly supervisors, Prof. Xiaodong Li and Assoc. Prof. Kai (Alex) Qin. I appreciative their extraordinary patience, unlimited support, professional integrity, and insightful feedback on my academic development.

I am also grateful for the research fund and travel grants I received from RMIT University and IEEE Computational Intelligence Society during my studies.

Throughout my Ph.D. studies, I met brilliant, dedicated, and yet humble and friendly colleagues in the RMIT AI group. In particular, I wish to name and thank Andy Song, Asad Mohammadi, Ayad Turky, Behrooz Ghasemishabankareh, Feng Xie, Jeffrey Chan, Jing Xie, Marco Tamassia, Minyi Li, Mohammad Nabi Omidvar, Simon Demediuk, Sven Schellenberg, and Vic Ciesielski. Their valuable suggestions on professional topics made my studies more productive.

I would like to extend my appreciation to my Iranian friends, with special mentions to Amin Sadri, Azadeh Gharineiat, Mohsen Laali, Reza Soltanpour, Shaahin Madani, Sargol Sadeghi, Sam Kharazmi, and Zhinoos Razavi Hesabi. Their companionship made this journey exceptionally pleasant and memorable for me.

Last but not least, I wish to express my sincerest gratitude to my lovely family for their unconditional support and continuous encouragement. I appreciate all the love I have ever received from my parents Ali Naghi and Marzieh, my brothers Mohammad and Masih, and my wife —and the best friend— Bahar.

Credits

Portions of the material in this thesis have previously appeared in the following publications:

1. B. Kazimipour, X. Li, and A. K. Qin. Initialization methods for large scale global optimization. In *IEEE Congress on Evolutionary Computation (CEC'13)*, pages 2750–2757. IEEE, 2013 (contribution in Chapter 4)

2. B. Kazimipour, X. Li, and A. K. Qin. A review of population initialization techniques for evolutionary algorithms. In *IEEE Congress on Evolutionary Computation (CEC'14)*, pages 2585–2592. IEEE, 2014b (contribution in Chapter 3)

3. B. Kazimipour, X. Li, and A. K. Qin. Effects of population initialization in differential evolution for large scale optimization. In *IEEE Congress on Evolutionary Computation (CEC'14)*, pages 2404–2411. IEEE, 2014a (contribution in Chapter 5)

4. B. Kazimipour, M. N. Omidvar, X. Li, and A. K. Qin. A novel hybridization of opposition-based learning and cooperative co-evolutionary for large-scale optimization. In *IEEE Congress on Evolutionary Computation (CEC'14)*, pages 2833–2840. IEEE, 2014d (contribution in Chapter 2)

5. B. Kazimipour, X. Li, and A. K. Qin. Why advanced population initialization techniques perform poorly in high dimension? In *Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'14)*, pages 479–490. Springer, Springer, 2014c (contribution in Chapter 6)

6. B. Kazimipour, M. N. Omidvar, X. Li, and A. Qin. A sensitivity analysis of contribution-based cooperative co-evolutionary algorithms. In *IEEE Congress on Evolutionary Computation (CEC'15)*, pages 417–424. IEEE, 2015 (contribution in Chapter 8)

7. M. N. Omidvar, B. Kazimipour, X. Li, and X. Yao. CBCC3 - A contribution-based cooperative co-evolutionary algorithm with improved exploration/exploitation balance. In *IEEE Congress on Evolutionary Computation (CEC'16)*. IEEE, 2016 (contribution in Chapter 8)

8. M. Yang, M. N. Omidvar, C. Li, X. Li, Z. Cai, B. Kazimipour, and X. Yao. Efficient resource allocation in cooperative co-evolution for large-scale global optimization. *IEEE Transactions on Evolutionary Computation*, PP(1):1–1, 2017 (contribution in Chapter 7)

9. B. Kazimipour, M. N. Omidvar, A. Qin, X. Li, and X. Yao. Bandit-based cooperative coevolution for tackling contribution imbalance in large-scale optimization problems. *Applied Soft Computing*, 76:265–281, 2019 (contribution in Chapters 9 and 10)

Contents

List of Figures

List of Tables

Abstract

Evolutionary algorithms are general purpose optimizers that have been shown effective in solving a variety of challenging optimization problems. In contrast to mathematical programming models, evolutionary algorithms do not require derivative information and are still effective when the algebraic formula of the given problem is unavailable. Nevertheless, the rapid advances in science and technology have witnessed the emergence of more complex optimization problems than ever, which pose significant challenges to traditional optimization methods.

The dimensionality of the search space of an optimization problem when the available computational budget is limited is one of the main contributors to its difficulty and complexity. This so-called *curse of dimensionality* can significantly affect the efficiency and effectiveness of optimization methods including evolutionary algorithms. This research aims to study two topics related to a more efficient use of computational budget in evolutionary algorithms when solving large-scale black-box optimization problems. More specifically, we study the role of population initializers in saving the computational resource, and computational budget allocation in cooperative coevolutionary algorithms. Consequently, this dissertation consists of two major parts, each of which relates to one of these research directions.

In the first part, we review several population initialization techniques that have been used in evolutionary algorithms. Then, we categorize them from different perspectives. The contribution of each category to improving evolutionary algorithms in solving large-scale problems is measured. We also study the mutual effect of population size and initialization technique on the performance of evolutionary techniques when dealing with large-scale problems. Finally, assuming uniformity of initial population as a key contributor in saving a significant part of the computational budget, we investigate whether achieving a high-level of uniformity in high-dimensional spaces is feasible given the practical restriction in computational resources.

In the second part of the thesis, we study the large-scale imbalanced problems. In many real world applications, a large problem may consist of subproblems with different degrees of difficulty and importance. In addition, the solution to each subproblem may contribute differently to the overall objective value of the final solution. When the computational budget is restricted, which is the case in many practical problems, investing the same portion of resources in optimizing each of these imbalanced subproblems is not the most efficient strategy. Therefore, we examine several ways to learn the contribution of each subproblem, and then, dynamically allocate the limited computational resources in solving each of them according to its contribution to the overall objective value of the final solution. To demonstrate the effectiveness of the proposed framework, we design a new set of 40 large-scale imbalanced problems and study the performance of some possible instances of the framework.

CHAPTER 1

Introduction

This chapter starts with a brief introduction to large-scale optimization that presents our primary motivation for conducting this research. Then, it lists the key research objectives and elaborates the methodology we follow to fulfill them. The chapter also clarifies our contribution to the field and achievements in this research. Finally, it ends by presenting the dissertation organization.

1.1 Motivation

Optimization problems are ubiquitous as we observe their traces in many scientific, engineering, and business related applications. Design optimization (aerospace engineering [Sobieszczanski-Sobieski and Haftka 1997, Wang et al. 2014]), utility maximization (microeconomics [Burkett 2006]), investment portfolio optimization (finance [Tapia and Coello Coello 2007]), well placement (petroleum engineering [Sarma et al. 2008]), autonomous navigation [Nearchou 1999, Manikas et al. 2007], and medical image processing [Lahanas 2004] are just a few examples of the optimization applications.

Recent advances in science and technology give rise to more complex optimization problems than ever [Mahdavi et al. 2014b, LaTorre et al. 2015]. The size of problems [Tang et al. 2007], the strong intercorrelation between their decision variables [Li 2014], and multiple contradicting objective functions [Miettinen 1999, Zitzler 1999] are some of the factors that contribute to the complexity of modern optimization tasks. The practical limitations in computational resources impose further difficulties in solving these problems in a timely manner [Jones et al. 1998].

There are two major approaches to solve an optimization problem; the mathematical models and the metaheuristics [Boussaïd et al. 2013]. A large body of the literature is devoted to the theoretically sound numerical optimization techniques, *e.g.,* gradient descent [Hestenes and Stiefel 1952, Snyman 2005]. These techniques usually begin from a single randomly generated potential solution which is iteratively improved towards a fitter solution by employing the gradient of the objective function as a guideline. The ultimate convergence of these mathematical models to global optima is guaranteed as long as their underlying assumptions are satisfied [Kiwiel 2001]. However, the mathematical models may result in sub-optimal solution(s) when the fundamental criteria (*e.g.,* the convexity of search space) cannot be met, the problem is too complex to model (*e.g.,* multiple intercorrelated variables), or the algebraic formulation of the problem is unknown [Conn et al. 2009].

A challenging type of optimization problems that regularly appear in practice is the *black-box* family. This term refers to optimization tasks where the analytic form of their objective functions are not available or their internal processes are too complex to be modeled accurately [Bäck 1996, Jansen et al. 2001]. Simulation optimization, for example, is a common type of black-box problems where the actual objective function is not formulated as an algebraic formula. Instead, synthetic simulators are used to evaluate the objective function of the simulators [Carson and Maria 1997]. In some other black-box problems in fields such as chemistry or biology, we do not have enough information about the mechanics of the objective function. Therefore, the only way to evaluate the objective function is through prolonged and expensive laboratory experiments [Li et al. 2013a].

The metaheuristic approach is more practical and advantageous in the aforementioned scenarios [Michalewicz and Fogel 2013, Larrañaga and Lozano 2001, Storn and Price 1997, Kirkpatrick et al. 1983]. For example, *Evolutionary Algorithms* (EAs[1]) and *Particle Swarm Optimization* (PSO) algorithms [Kennedy and Eberhart 1995, Kennedy 2011] are two broad categories of bio-inspired derivative-free optimization techniques that are widely used to solve black-box problems [Lucasius and Kateman 1991]. These stochastic algorithms typically start with a population of candidate solutions and iteratively refine the candidates with the hope to produce fitter solutions regarding the objective function. Although there is rarely any theoretical guarantee for the convergence of metaheuristics, they have achieved promising results in solving a wide range of real-world applications.

While the number of complex and challenging optimization problems grows every day, the traditional optimizers usually fail to solve them effectively and efficiently [Dolan et al. 2004]. The search space dimensionality (*i.e.,* the domain of the function to be optimized) is one of the main factors which makes solving these recent problems very challenging [Weise et al. 2012]. This factor contributes to the complexity of large-scale problems in three ways:

1. The volume of the search space expands exponentially as the problem's dimensionality increases linearly. For instance, the number of potential solutions for a binary optimization problem with 100 decision variables is $1.12E{+}15$ times larger than a 50-dimensional problem. Note that the growth factor is immensely greater in the case of continuous problems.

2. The evaluation of high-dimensional problems is rather long and expensive. Some aerospace design tasks, for example, may need hours or even days of wind tunnel laboratory work for evaluation of each single candidate solution. Bearing in mind that the expensive evaluation is a substantial issue in large-scale optimization since the solvers demand numerous evaluations as the search spaces of these problems are massive (referring to the issue mentioned earlier).

3. The properties of the search landscape may change as the number of variables increases. This means some optimization techniques that theoretically could solve similar problems in low-dimensional space may perform poorly as the number of decision variables increases. The Rosenbrock function is a classic example that exhibits a unimodal search landscape in a two-dimensional space but becomes multimodal in higher dimensions [Shang and Qiu 2006]. For that reason, many optimizers that can quickly solve a two-dimensional unimodal Rosenbrock function may face difficulties to solve its high-dimensional multimodal expansions. Another well-known example is Rastrigin function which its number of local optima is an exponential function of

[1]Some researchers may use the term EAs loosely as a reference to the broader family of metaheuristics.

its number of dimensions [Mühlenbein et al. 1991] (*i.e.,* the number of local optima is D^{11} where D is the dimensionality of the problem).

Researchers have proposed several techniques for solving large-scale optimization problems. Most of these techniques fall into one of the following categories:

1. dimensionality reduction algorithms that try to address the first issue mentioned earlier [Shan and Wang 2010b],
2. function approximation and surrogate models that make the function evaluation faster (addressing the second issue) [Queipo et al. 2005, Regis 2013, Chen et al. 2012, Ren et al. 2019],
3. smart sampling [Rahnamayan and Wang 2008b; 2009, de Melo and Botazzo Delbem 2012b], local searches [Molina et al. 2010, LaTorre et al. 2013], and memetic algorithms [Chen et al. 2011b] that try to spend a bigger portion of computational budget on the local searches instead of global exploration,
4. decomposition-based techniques that split large-scale problems into smaller subproblems and solve them almost separately [Chen and Tang 2013, Li 2014, Omidvar et al. 2014b],
5. self-adaptive [Qin and Suganthan 2005, Yang et al. 2008c] and hyper-heuristics [Burke et al. 2013] that adaptively update the optimizer's parameters setting to achieve the best results, and
6. hybrid techniques that combine one or more of the above approaches [LaTorre et al. 2013, El-Abd 2014].

Although the strategies mentioned above improve the traditional techniques, they are still prone to the *curse of dimensionality*. For example, the efficiency and effectiveness of dimension reduction techniques usually drop as the number of variables increases. In the case of function approximation and hyper-heuristic approaches, the employed statistical techniques and machine learning algorithms need lots of training samples to model high-dimensional search spaces. Producing such massive datasets demands many objective function evaluations which are impractical in various real-world applications. In many approaches, such as local searches and memetic algorithms, the computational budget is spent uniformly on optimizing all decision variables regardless of their contribution to the quality of the final solution. Finally, the effectiveness of the decomposition-based techniques is strongly correlated with the optimality of their decomposer algorithm. The decomposers also consume a portion of the limited computational budget, which sometimes results in early termination of the optimization process.

The practical limitation in computational resources is the Achilles heel of all the existing large-scale optimizers, regardless of the approach they took. In many real-world applications, the available computational budget is not enough to find the global optima of a large-scale problem. Even in the rare cases that finding the global optimum is feasible in the given time window, improving the efficiency of the algorithms in terms of minimizing computational costs is invaluable. The ultimate goal of this research project is to study and develop a number of techniques that lead to the more economical use of limited computational budget when dealing with large-scale black-box optimization problems.

1.2 Research Objectives

To study the effectiveness of the existing computational cost reduction strategies and propose some new techniques, we narrowed down the domain of this research to two major topics: 1) the effect of population initialization on the computational costs, and 2) effective resource allocation in cooperative coevolutionary algorithms. The central goals that we will attain are:

1. To collect, study, survey, and categorize the published scientific articles on population initialization techniques that have been used in metaheuristics.
2. To assess the quality of initial populations generated by different techniquesas as the number of decision variables grows.
3. To study the mutual effects of population initialization and optimizer parameters on the performance of metaheuristics when the computational resources are limited.
4. To analyze and improve the robustness of the contribution-aware cooperative coevolutionary techniques in dealing with imbalanced problems.
5. To design and implement a comprehensive set of large-scale imbalanced problems and benchmark the contribution-aware techniques on the proposed problem suite.
6. To formulate the economical resource allocation problem as a dynamic multi-armed bandit task, and propose a general cooperative coevolutionary framework to tackle large-scale imbalanced problems more efficiently.

As discussed above, the focus of the first three objectives is on using more effective population initialization techniques when the computational budget is limited, whereas the last three goals try to address the waste of computational budget on less critical components of large-scale imbalanced problems.

1.3 Methodology

We employ a combination of analytical and empirical methods to achieve the objectives mentioned above. We leverage on standard mathematical formalism to formulate the problem definitions as well as conventional pseudocodes to elaborate the algorithmic solutions. In the empirical studies, we use a number of widely-used large-scale optimization benchmarks to assess the performance of the existing methods and our proposed algorithms. Whenever required, we expand the standard benchmark sets to create more use cases with different features and characteristics. These extended test suites are extremely helpful in understanding the behavior of the algorithms in the less-explored scenarios. We evaluate the statistics of the final solutions in multiple independent trials to compare the performance of our proposed optimizers with the baselines, the counterpart techniques, and the state-of-the-art algorithms. Whenever applicable, other metrics besides the final objective values are also taken into account. For example, we use discrepancy measures to compare the uniformity of the initial populations or the distribution of the selected subproblems to evaluate the effectiveness of the budget allocation systems in contribution-aware cooperative coevolutionary algorithms. Finally, we extensively use parametric and nonparametric significance tests to avoid making arbitrary assumptions about the statistical significance of the experimental results.

1.4 Our Contributions

This study makes contributions to the field of large-scale black-box optimization from two different but related perspectives: cost-effective population initialization in high-dimensional spaces, and efficient budget allocation when dealing with large-scale imbalanced problems. In particular, the key contributions are:

1. A comprehensive study followed by a multi-facet categorization of the population initialization techniques that have been used in population-based metaheuristics.
2. The study of the mutual effects of problem dimensionality and population size on the effectiveness of high-dimensional initial populations generators.
3. The investigation of the reason behind the poor performance of uniform population initialization techniques when adopted to solve large-scale problems.
4. The development of a comprehensive set of large-scale imbalanced problems to be used in budget allocation research.
5. The benchmarking of the existing contribution-aware cooperative coevolutionary algorithms using the large-scale imbalanced problem set.
6. The improvement of budget allocation capabilities of the previously proposed contribution-aware cooperative coevolutionary algorithms by enhancing their exploration-exploitation balance in the component space.
7. The deign, development, and anlaysis of a bandit-based cooperative coevolutionary framework for effective computational budget allocation.

1.5 Thesis Organization

The rest of this dissertation is organized as follows. We devote the next chapter to key definitions and background material about optimization in general and population-based metaheuristics for large-scale black-box optimization in specific. In Chapter 2, we point out the potentials and the pitfalls of the existing optimization techniques. After the Background and Literature Review chapter, we present two contribution parts each of which consists of several chapters. We dedicate the first part to study the population initialization algorithms and their potential role in advancing large-scale metaheuristics where the computational budget is limited. This part includes four chapters:

- Chapter 3 surveys the existing population initialization techniques. It also categorizes them into multiple classes from three unique perspectives. This chapter sets a foundation for the next chapters of the first part.
- Chapter 4 empirically studies the performance of population initialization techniques as the dimensionality of the problem grows. Given a fixed computational budget, we investigate whether alternative population initialization techniques can improve the outcomes of the conventional techniques.
- Chapter 5 compares the mutual effect of population initialization techniques and population size on the quality of the final solutions. The goal of the numerical experiments in this chapter is to identify the main contributing factor, either the size of the initial population or the way we generate it.

- Chapter 6 provides further insights on the uniformity of the uniform population generators. Assuming the uniformity of the initial population as a critical contributing factor in the performance of metaheuristics when solving a black-box problem, we aim to investigate if a high degree of uniformity is achievable given the limitations in the computational budget.

In the second part, we study large-scale imbalanced problems and propose some novel techniques to address the budget allocation issue in such scenarios. The last part contains the following four chapters:

- Chapter 7 formally defines the contribution imbalance problem and reviews the contribution-aware cooperative coevolutionary techniques that have been designed to address the budget allocation problem when the imbalance contribution exists.
- Chapter 8 investigates the exploration versus exploitation balance in two well-known contribution-aware techniques. In this chapter, we improve these techniques by proposing a few algorithmic modifications.
- Chapter 9 proposes an extensive set of large-scale imbalanced problems and then benchmarks almost all existing contribution-aware cooperative coevolutionary techniques using the proposed testbed.
- Chapter 10 briefly reviews the multi-armed-bandit as a classic, yet popular, approach to address dynamic resource assignment problems. Then, it formulates the budget allocation problem in imbalanced optimization tasks as a multi-armed bandit task. Finally, it proposes a new generic cooperative coevolutionary algorithm that leverages multi-armed bandit techniques to solve the imbalanced problems effectively.

Finally, Chapter 11 concludes the thesis with a summary of the major findings and a list of possible future work.

CHAPTER 2

Background and Literature Review

This chapter contains a general background and a number of important definitions that we use throughout the dissertation. The primary objective of this chapter is to provide a big picture of the advances, potentials, and challenges of large-scale optimization. Having such discussions helps us to set a firm foundation and then narrow down the topic to frame the research objectives.

In this chapter, we first formally define the key concepts of large-scale black-box optimization. Then, we briefly discuss different forms of such problems and explain which aspects will be studied in this research. Finally, we review the major evolutionary computation frameworks such as decomposition-based, nondecompositional, and hybrid techniques that have been used to overcome the curse of dimensionality in optimization. Note that we deliberately postpone the detailed explanation of some of the less general background materials, *e.g.,* **discrepancy measures** and **multi-armed bandit solvers**, to the most relevant chapters. This helps us to better manage the size and coherence of this chapter.

2.1 Optimization Problems

Optimization is the process of searching a solution space to find an input of a function that triggers an output with the highest or lowest possible value. Without loss of generality, we can define a minimization problem in the following form.

Definition 1 (Minimization Problem)**.** A minimization problem is defined as to finding a D-dimensional vector $\overset{*}{\mathbf{x}}$ such that $\forall \mathbf{x} \in \mathbb{R}^D \; f(\overset{*}{\mathbf{x}}) \leq f(\mathbf{x})$. More formally, a minimization problem is expressed as:

$$\begin{array}{rl} \min & f(\mathbf{x}), \\ \text{where} & \mathbf{x} \in \mathbb{R}^D. \end{array} \tag{2.1}$$

In the above equation, $f\colon \mathbb{R}^D \to \mathbb{R}$ is called the *objective function* and $\mathbf{x} = \langle x_1, \ldots, x_D \rangle$ is a D-dimensional *decision vector* where each x_d is a *decision variable*. Note that hereafter, the $\overset{*}{\mathbf{x}} = \arg\min f(\mathbf{x})$ denotes the global optimum of function f. In some cases, it is possible to have more than one global optimum for one objective function. For example, there could be an i and a j such that $\overset{*}{\mathbf{x}}_i \neq \overset{*}{\mathbf{x}}_j$ but $f(\overset{*}{\mathbf{x}}_i) = f(\overset{*}{\mathbf{x}}_j) \leq f(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^D$. The functions that have such property are referred to as *multi-modal* optimization problems. In the followings, we briefly review some of the other possible attributes of optimization problems and discuss which of them we are interested in.

Constrained Problems: In the most general form of optimization tasks that is formulated in Equation (2.1), the search space is only box-bounded. This implies that $\overline{X}_d \leq x_d \leq \underline{X}_d$ for all $d \in \{1, \ldots, D\}$, where $\overline{X}_d$ and $\underline{X}_d$ are the lower and upper bounds of the d^{th} decision variable, respectively. In other words, the domain of the function f is limited to only a subset of $\mathbb{R}^D$. Therefore, any point outside this bounded space is not a feasible solution to the problem. For example, if a number of variables refer to some physical measurements such as length or mass, they cannot accept negative values.

In many real-world optimization tasks, a number of other linear or nonlinear constraints may also be applied to the problem [Bondarenko et al. 1999]. The constrained counterpart of the minimization problem in Equation (2.1) can be formally defined as follows:

Definition 2 (Constrained Minimization Problem)**.**

$$\begin{array}{rll} \min & f(\mathbf{x}), & \\ \text{subject to}: & g_k(\mathbf{x}) = 0 & \forall k \in \{1, \ldots, K\}, \\ & h_l(\mathbf{x}) \geq 0 & \forall l \in \{1, \ldots, L\}, \end{array} \tag{2.2}$$

where $\mathbb{G} = \{g_k(\mathbf{x})\}_1^K$ and $\mathbb{H} = \{h_l(\mathbf{x})\}_1^L$ are the sets of *equality* and *inequality* constraint functions, respectively.

Regarding the popularity of constrained problems in science and technology, many advanced techniques have been developed to address these tasks directly or convert them to unconstrained problems [Coello Coello 2002, Mezura-Montes and Coello Coello 2011]. Technically, the so-called *discrete* optimization functions are special cases of constrained problems where the domain of decision variables is restricted to a finite or infinite number of discrete values (*e.g.,* integers). Many classic optimization problems such as shortest path and knapsack are discrete constrained problems. To keep the domain of this research manageable, we assume that an appropriate technique effectively handles the constraints. This assumption is widespread in large-scale optimization studies [Li et al. 2013a].

Multiobjective Problems: Multiobjective optimization tasks that are widely found in practice are those problems that involve two or more objective functions.

Definition 3 (Multiobjective Optimization Problem)**.** In mathematical terms, a multi objective minimization problem is expressed as:

$$\min \left(f_1(\mathbf{x}), \ldots, f_K(\mathbf{x}) \right), \tag{2.3}$$

where $K > 1$ is the number of objectives. These functions usually contradict with each other such that the optimum solution of f_i might not be even close to the optimum solution of f_j, assuming $i \neq j$. Therefore, the goal here is to find a set of trade-off solutions which approximates the *Pareto-front*.

There is a vast amount of literature on solving multiobjective problems using metaheuristics [Zitzler 1999, Coello Coello et al. 2002]. One of the standard practices is *scalarization* which converts a multiobjective optimization task into a single-objective problem. For example, a liner scalarizing can be formulated as:

$$f(\mathbf{x}) = \sum_{k=1}^{K} c_k \cdot f_k(\mathbf{x}), \tag{2.4}$$

where $c_k > 0$ are the coefficients of the objectives which can be constant or variable. Therefore, instead of dealing with a K-dimensional objective space as in Equation (2.3), we need to solve one single function in a 1-dimensional objective space as formulated in Equation (2.4). Setting different values for coefficients will change the search space and hence the final solutions. In the entire of this thesis, we assume that the given optimization problem is either single-objective by nature or converted to a single-objective problem using an arbitrary scalarizing algorithm.

Black-Box Problems: Originally, the term black-box refers to situations when we have absolutely no knowledge about their internal processes. Therefore, the algebraic formulation of objective function f, and thus its derivatives, are unavailable. However, we can still query the function by feeding an input vector $\mathbf{x}$ and record the returned output $f(\mathbf{x})$. In such a case, the derivative-based optimizers are impractical [Conn et al. 2009].

We may expand the traditional definition of black-box problems to any optimization task where the accurate analysis of the mathematical formulation can be practically infeasible or computationally expensive. For example, we may be able to develop an artificial simulation of the complex physical phenomenon in the human brain; thus technically they are not entirely black-box. However, solving such problems using the conventional mathematical techniques could be practically challenging, if possible, since the simulation is a complex and still a rough approximation of the real process. Many practitioners prefer to deal with this type of problems similar to black-box tasks: as if there is no useful information about the internal processes.

In this dissertation, we study and develop general-purpose derivative-free algorithms which do not make strong assumptions about the internal process of the objective functions [Rios and Sahinidis 2013]. Therefore, we can safely apply the studied techniques to any black-box task. Note that, we do not deny the importance of the domain knowledge about the problem. In some cases, such information can help the practitioners to improve the adopted algorithms. Despite this, we develop algorithms that can still be applicable even if there is very little knowledge about the underlying process of the given problem.

Large-Scale Problems: A large-scale optimization problem is nothing more than a regular optimization task with many decision variables (*i.e.,* a large D in Equation (2.1)). Although there is considerable literature about large-scale optimization, there is no universally accepted lower bound on the dimensionality of large-scale problems. Therefore, one may argue that if 100-dimensional problem should be considered as a large-scale problem or not.

The main feature that all large-scale problems have in common is their high complexity that is enforced by their dimensionality. The lack of enough computational resources to solve such complex problems on time is the main factor that prevents the traditional techniques to solve problems with such great complexity. Since large-scale optimization is the central topic of this dissertation, we will discuss it further in Section 2.3.

Up to this point, we provided a general definition of large-scale black-box optimization problems that we study in the rest of this dissertation. As mentioned earlier, we assume that the given tasks are, or already converted to, unconstrained single-objective continuous problems. In particular, we are interested in solving large-scale black-box problems in the face of limited computational budget. In the next section of this chapter, we briefly review the metaheuristics that have been previousely developed to tackle such problems.

2.2 Metaheuristics

There are two major approaches to solving an optimization problem; the exact techniques (*a.k.a.* mathematical models) and the approximation methods (*a.k.a.* metaheuristics) [Boussaïd et al. 2013]. Exact techniques are those algorithms that can find the true optimum solution of a given problem as long as all of their underlying assumptions are satisfied. These theoretically sound techniques have been used to successfully solve a variety of optimization problems [Hestenes and Stiefel 1952, Snyman 2005]. Convex programming [Bertsekas et al. 2003] including linear programming [Beasley 1996], integer programming [Nemhauser and Wolsey 1988, Schrijver 1998], and dynamic programming [Bellman 1956] are some of the well-known subcategories of exact optimization techniques.

Applying exact techniques to many real-world problems is infeasible due to the unsatisfied assumptions [Kiwiel 2001]. In many other cases, finding the exact model or even investigating whether the assumptions are met or not, demand a lot of time and effort. In such scenarios, practitioners usually convert the original problem to a simpler problem that satisfies the underlying assumptions. This process is usually referred to as *problem relaxation*. Although the relaxed problem can be solved using the available exact techniques, there is no guarantee that the exact solution to the relaxed problem remains the optimum solution to the original problem. Accordingly, the relaxation process may help to make the original problem easier to solve. However, it can also degrade the quality of solutions found, some of which might not be even feasible solution for the original problem. [Conn et al. 2009].

In contrast with the first group, approximation techniques make very few assumptions about the given problem. This attribute makes them general purpose algorithms that can be applied to virtually any optimization task. Therefore, there is no need to invest time and effort to investigate the satisfaction of the assumption (as they rarely make any significant assumption) or formulate the problem in some specific form (since they do not need such formulations) [Bäck 1996, Jansen et al. 2001].

The main drawback of the approximation techniques is that in many cases there is no guarantee to find the optimal solution. In practice, though, researchers show that the approximation techniques can find good enough solutions in a manageable timeframe. This makes these techniques very popular in practice despite their theoretical limitations. In fact, practitioners can choose between the tools that provide some approximation of the original solution and the techniques capable of finding an optimal solution to the relaxed problems [Kirkpatrick et al. 1983].

One of the most prominent families of the approximate optimizers belongs to metaheuristics [Michalewicz and Fogel 2013]. These optimizers are essentially search algorithms that can provide sufficiently good solutions to optimization problems even when the information about the search landscape is incomplete, or the available computational budget is restricted. These algorithms are extremely useful when the solution set is too large to be thoroughly sampled.

Evolutionary Computation (EC) techniques are a group of stochastic metaheuristics inspired by nature. Technically, they are a subgroup of population-based trial and error problem solvers. This means, EC techniques usually start with an initial set of potential solution generated blindly or based on some domain knowledge. Then, by iteratively applying evolutionary operators on the current population, they generate another set of potential solutions called offspring population. Then, a subset of these two populations is selected based on some criteria (usually guided by a fitness function) and delivered to the next iteration (*a.k.a.* generation). This procedure is repeated until a termination criterion, *e.g.,* the maximum number of function calls or minimum solution quality, is met.

The fitness function which is used to guide the algorithms can be as simple as the objective function in a maximization problem, or its negation in a minimization task. Therefore, the ultimate goal of an EC algorithm to maximizing the fitness value can be easily translated into solving the original optimization problem.

The large family of EC techniques is further divided into several subcategories such as Evolutionary Algorithm (EA) [Holland 1975], Genetic Algorithm (GA) [Goldberg 1989], Particle Swarm Optimization (PSO) [Kennedy and Eberhart 1995, Kennedy 2011], Ant Colony Optimization [Dorigo et al. 1996], Differential Evolutionary (DE) [Price et al. 2006], Estimation of Distribution Algorithms (EDA)[Larrañaga and Lozano 2001], etc. Though all of these subsets share the common theme of the EC model, they may differ in some details. For example, GAs' operators are inspired by biological evolution (*e.g.,* reproduction, mutation, recombination, and natural selection) whereas PSO is essentially a simulation of the group behavior of fish schools or bird flocks.

2.3 Large-Scale Optimization

2.3.1 Definitions

As mentioned in 2.1, the term large-scale refers to high-dimensional optimization problems with many decision variables[1]. Notably, the notion of large-scale can be subjective. In this context, we consider a problem large when the existing optimizers show scalability issues given a certain amount of computational budget. In particular, we focus on problems with at least $1,000$ decision variables which is the current field standard [Li et al. 2013a, Omidvar et al. 2015].

Definition 4. A large-scale box-bounded single-objective minimization is defined as:

$$\begin{array}{rll} \min & f(\mathbf{x}), & \\ \text{where :} & \overline{X}_d \leq x_d \leq \underline{X}_d & \forall d \in \{1,\ldots,D\}, \\ \text{and} & D \geq 1000. & \end{array} \tag{2.5}$$

Generally speaking, three major directions have been followed in the literature to reduce the adverse effect of dimensionality on the performance of metaheuristics. One approach attempts to decompose problems into smaller pieces so that existing algorithms can handle them individually. To maximize the performance of these *decomposition-based* techniques, the original large-scale problem should be carefully divided into ideally isolated subproblems. In the case of black-box problems, the links between the variables are unknown to the practitioners. Therefore, effective decomposition techniques must be employed to reveal these interconnections. We discuss the decomposition-based techniques in Section 2.5.

Another approach aims to enhance the available optimizers by the aid of different techniques such as advanced initialization [Mahdavi et al. 2016a], intelligent sampling [LaTorre et al. 2012], memetic algorithms [Molina et al. 2010], and adaptation [Brest and Maučec 2011]. We explain these *nondecompositional* techniques in Section 2.4.

The last approach is simply the combination of the other two. These techniques plug nondecompositional optimizers into the decomposition-based techniques to create a more effective hybrid [Peng and Wu 2018]. We devote Section 2.6 to this approach.

Before we describe the aforementioned large-scale optimization approaches in more details, it is essential to review some of the real-world high-dimensional applications (see

[1]Problems with high-dimensional objective space are usually referred as many-objective problems. This topic is out of the scope of this research.

Subsection 2.3.2). Indeed, this review exhibits some of the potential applications of this research. Likewise, we concisely summarize the synthetic high-dimensional benchmark testbeds that were specially designed to assess the efficiency of the large-scale optimizers (see Subsection 2.3.3).

2.3.2 Real-world Large-Scale Optimization Problems

As mentioned in Equation (2.5), the current academic standard for large-scale optimization is problems with 1,000 dimensions. However, in some rare cases, the dimensionality of the solved problems are as high as 800,000 variables [Sousa et al. 2016]. There are also a few published studies in the literature of metaheuristics that used the term large-scale while referring to tasks with less 1,000 decision variables.

One of the recent applications of bio-inspired optimizers is the Social Network Analysis where scientists study the complex graphs generated by human relationships. For example, metaheuristics can be used to select a small subset of nodes from a large-scale online social network that maximizes the influence propagation (*a.k.a.* Networks Influence Maximization) [del Puente 2017]. Solving such problem that can grow to millions of node is of great importance for enabling viral campaign in online spaces [Sankar et al. 2016]. It is proved that influence maximization is a Sharp-P-hard problem [Chen et al. 2010, Wang et al. 2012a].

The applications of metaheuristics in Big Data is much wider than the social media analysis [Cheng et al. 2013, Bhattacharya et al. 2016, Cheng et al. 2016]. According to [Sanchita and Anindita 2016], while traditional search algorithms are too slow for finding an acceptable solution on time, metaheuristics have shown to be fast and promising. These techniques are particularly useful as they require little information to search such enormous and complex search spaces effectively. Clustering high-dimensional text datasets [Lu et al. 2009; 2011, Karol and Mangat 2013], Web mining [Pal et al. 2002], instance selection [García and Herrera 2009, García-Pedrajas et al. 2010, García-Pedrajas 2011], subspace clustering [Lin et al. 2014], and prototype selection [García et al. 2008] are just a few applications of metaheuristics in the handling and analyzing Big Data.

A closely related topic that metaheuristics have been successfully applied onto is the data mining and knowledge discovery in huge datasets [Freitas 2009]. Since the traditional knowledge discovery techniques suffer from the curse of dimensionality, different metaheuristics (especially cooperative coevolution algorithms) are adopted to lessen the adverse effect of dimensionality [García-Pedrajas and de Haro-García 2012, Xue et al. 2016]. For example, an evolutionary approach to feature selection [Hong and Cho 2006, Robbins et al. 2007, Li et al. 2013b, Meena and Ibrahim 2016] and simultaneous feature and sample selection [Derrac et al. 2012, GarcíA-Pedrajas et al. 2013, Pérez-Rodríguez et al. 2015] are shown to be useful when facing large-scale datasets. Besides, metaheuristics are also successfully employed to improve machine learning algorithms in different ways. For instance, it has been shown that nature inspired optimizers can be used to evolve an ensemble of artificial neural networks [García-Pedrajas et al. 2005; 2003], predict complex structures [Bahmann and Kortus 2013], create effective rule-based systems [Srinivasan and Ramakrishnan 2011], and classify high-dimensional patterns [Ishibuchi and Namba 2004].

The route networks are globally expanding very fast which makes the routing and scheduling problems more complex than ever. These multiobjective optimization problems typically have vast and complex search landscapes [Vlahogianni 2015]. Metaheuristics are repeatedly reported to be suitable choices [Shimamoto et al. 2010]. For example, these algorithms are adopted for energy saving scheduling in railways [Chevrier et al. 2013], optimizing bicycle renting models [Chira et al. 2014], bus routing [Pattnaik et al. 1998],

highway alignment [Jong and Schonfeld 2003], and optimal deployment of many emergency response units in Athens transportation network [Geroliminis et al. 2011]

The emerging large-scale sensor networks are another type of complex graphs that we know as the Internet of Things. These networks have numerous medical and environmental applications such as habitat monitoring, smart factory instrumentation, intelligent transportation, disaster management, and remote surveillance [Kulkarni and Venayagamoorthy 2011, Kulkarni et al. 2011]. Regarding these wide applications, sensor networks are expanding at a faster pace than the route networks world-while. A number of sensor network optimization problems are addressed by metaheuristics such as network localization [Gopakumar and Jacob 2008], optimal transmission power allocation [Hong and Shiu 2007], cluster formation [Guru et al. 2005], optimal power scheduling [Wimalajeewa and Jayaweera 2008], and dynamic sensor management [Veeramachaneni and Osadciw 2004].

Other large-scale applications of metaheuristics that are worth mentioning are: engineering design [Shan and Wang 2010a, Akay and Karaboga 2012, Yi et al. 2013], routing [Mei et al. 2011, Mingming et al. 2011, Mei et al. 2014a;b], scheduling [Xhafa and Abraham 2008, Marchiori and Steenbeek 2000], modeling [Zamuda et al. 2011], telecommunication [Gutiérrez et al. 2011], cloud computing [Ai et al. 2011], chemistry [Kononova et al. 2008], sociology [Huang et al. 2012, Atay et al. 2017], biology [Villaverde et al. 2012, Thomas and Jin 2014, He et al. 2016], and bio-mechanics [Koh et al. 2009].

Note that, the main focus of this work is on scaling metaheuristics when solving black-box problems with many continuous variables. Therefore, one can directly and easily apply the findings and proposed techniques to problems with similar nature. However, practitioners may need to adjust the proposed algorithms when adopting them to solve discrete or multiobjective optimization tasks.

2.3.3 Synthetic Benchmarking Problems

To the best of our knowledge, the Benchmark Functions for the IEEE CEC 2008 Special Session and Competition on Large Scale Global Optimization (*a.k.a.* CEC'08 LSGO benchmark suite) [Tang et al. 2007] is the first attempt to proposing a set of optimization problems with the aim of providing a suitable evaluation platform for testing and comparing large-scale optimization algorithms. The suite included three separable (*i.e.,* Sphere, Rastrigin's, and Ackley's functions) and four fully-nonseparable (*i.e.,* Schwefel's Problem 2.21, FastFractal DoubleDip, Rosenbrock's, and Griewank's) functions[2]. The 100, 500, and 1,000–dimensional instances of these problems were implemented in C, Java and Matlab languages. The maximum number of objective function evaluation was fixed to $5000 \times D$ where D is the dimensionality of the problem. Table 21 summarizes this set.

The CEC'08 benchmark suite was successful in building a scalable set of benchmark functions in order to promote research in the field of large-scale global optimization. However, the set suffers from a few significant shortcomings. Firstly, the number of functions in this suite was limited (only seven problems). Therefore, it was difficult to statistically compare three or more optimization algorithms using this limited number of problem instances. Secondly, the included problems were either fully-separable (no interaction between any pair of variables) or fully-nonseparable (with a strong interaction between any possible pair of variables). As a result, the suite was expanded to include more problems with more diverse properties. The partially-separable functions were included in CEC'10 [Tang et al. 2009] for the first time.

[2]We will formally define the *separability* in Section 2.5.

Table 21: IEEE CEC'08 LSGO Benchmark Set

Problem	**Basis Function**	**Modality**	**Separability**
f_1	Sphere Function	Unimodal	Separable
f_2	Schwefel's Function 2.21	Unimodal	Nonseparable
f_3	Rosenbrock's Function	Multimodal	Nonseparable
f_4	Rastrigin's Function	Multimodal	Separable
f_5	Griewank's Function	Multimodal	Nonseparable
f_6	Ackley's Function	Multimodal	Nonseparable
f_7	DoubleDip Fast Fractal	Multimodal	Nonseparable

The IEEE CEC 2010 Special Session and Competition on Large Scale Global Optimization (*a.k.a.* CEC'10 LSGO set) consisted of 20 problems each of which had 1,000 variables [Tang et al. 2009]. For the first time, a new category of problems called m-nonseparable (*a.k.a.* partially-separable) functions was also included in the set. By definition, an m-nonseparable problem is a function which at most m of its parameters are not independent. Generally speaking, CEC'10 included five categories of problems:

1. three fully-separable problems,
2. five partially separable problems each of which consisted of 50 coupled variables and 950 independent variables,
3. five modular problems each of which consisted of ten 50–dimensional nonseparable components and 500–dimensional fully-separable problems,
4. five modular problems each of which is comprised of 20 50–dimensional nonseparable components,
5. two fully-nonseparable functions.

All the 20 problems were created based on six basis functions, and it was recommended to keep the number of function evaluation less than 3,000,000. This limit was 2,000,000 units less than the proposed budget for CEC'08 problems having the same dimensionality. As a result, solving the new set is more challenging than its predecessor. Table 22 summarizes this suite.

The main advantage of the CEC'10 benchmarks over the previous set is that it represents the modular feature that many real-world problems exhibit. Nevertheless, advances in this domain signal the need to revise and extend the existing benchmark suite once again. For example, the Differential Grouping algorithm could detect the grouping structure of the majority of the problems with 100% accuracy [Omidvar et al. 2014a]. Besides, these functions were not adequate to study problems having components with imbalance contributions.

The IEEE CEC 2013 Special Session and Competition on Large Scale Global Optimization (CEC'13 LSGO) benchmarks were specially developed to better represent the features of a broader range of real-world problems, as well as posing new challenges to the existing optimizers, especially to decomposition-based algorithms [Li et al. 2013a]. The new types of problems that were included in this suite are:

1. problems with nonuniform component sizes,
2. problems with unequal components' coefficients,

Table 22: IEEE CEC'10 LSGO Benchmark Set

Problem	**Basis Function**	**Separable**	**Nonseparable**
f_1	Elliptic Function	1000	0
f_2	Rastrigin's Function	1000	0
f_3	Ackley's Function	1000	0
f_4	Elliptic Function	950	1×50
f_5	Rastrigin's Function	950	1×50
f_6	Ackley's Function	950	1×50
f_7	Schwefel's Problem 1.2	950	1×50
f_8	Rosenbrock's Function	950	1×50
f_9	Elliptic Function	500	10×50
f_{10}	Rastrigin's Function	500	10×50
f_{11}	Ackley's Function	500	10×50
f_{12}	Schwefel's Problem 1.2	500	10×50
f_{13}	Rosenbrock's Function	500	10×50
f_{14}	Elliptic Function	0	20×50
f_{15}	Rastrigin's Function	0	20×50
f_{16}	Ackley's Function	0	20×50
f_{17}	Schwefel's Problem 1.2	0	20×50
f_{18}	Rosenbrock's Function	0	20×50
f_{19}	Schwefel's Problem 1.2	0	1000
f_{20}	Rosenbrock's Function	0	1000

3. problems with overlapping subproblems.

The first two types of problems represent scenarios that some components of a given task are more difficult to solve (*i.e.,* demand more computational budget) or have a more substantial influence on the fitness of the solution. The category of overlapping problems is a particular case of nonseparable problems where some variables have interaction with some other variables where there is no single variable or group of variables that can be identified as an isolated component. Therefore, these problems cannot be categorized as partially-separable functions although their interaction topology is not a complete graph as is the case for fully-nonseparable problems.

Solving the CEC'13 problem set is more challenging than the older suites. In particular, the previously proposed decomposition techniques usually fail to decompose the overlapping and imbalanced problems with the same accuracy they could achieve on the CEC'10 cases [Sun et al. 2019]. Although the decomposition task is out of the scope of this research, we extensively use the CEC'13 functions to compare the performance of different algorithms. There are two reasons to support such a decision. Firstly, the CEC'13 is the most recent available benchmark suite. Secondly, the imbalanced problems that are very common in the real-world application are only available in this set of functions. When dealing with this type of problems, effective computational budget allocation becomes crucial. Table 23 summarizes this set.

Besides all the benefits that CEC'13 brings to our research, it has some limitations as well. For example, some sources of imbalance are not represented in this set, and the number of imbalanced problems is not adequate for the statistical comparison of three or

Table 23: IEEE CEC'13 LSGO Benchmark Set

Problem	Basis Function	Contribution	Separability
f_1	Elliptic Function	Balanced	Separable
f_2	Rastrigin's Function	Balanced	Separable
f_3	Ackley's Function	Balanced	Separable
f_4	Elliptic Function	Imbalanced	Partially and Fully Separable
f_5	Rastrigin's Function	Imbalanced	Partially and Fully Separable
f_6	Ackley's Function	Imbalanced	Partially and Fully Separable
f_7	Schwefel's Problem	Imbalanced	Partially and Fully Separable
f_8	Elliptic Function	Imbalanced	Partially Separable
f_9	Rastrigin's Function	Imbalanced	Partially Separable
f_{10}	Ackley's Function	Imbalanced	Partially Separable
f_{11}	Schwefel's Problem 1.2	Imbalanced	Partially Separable
f_{12}	Elliptic Function	Balanced	Overlapping Nonseparable
f_{13}	Rastrigin's Function	Balanced	Overlapping Nonseparable
f_{14}	Ackley's Function	Balanced	Overlapping Nonseparable
f_{15}	Schwefel's Problem 1.2	Balanced	Fully Nonseparable

more algorithms. In Chapter 9, we significantly expand CEC'13 benchmark set to cover more scenarios.

2.4 Nondecompositional Techniques

As mentioned earlier, we categorize the large-scale optimizers into three major groups: nondecompositional algorithms that try to scale-up the available metaheuristics to handle high-dimensional problems, the decomposition-based techniques that divide a large-scale problem into its smaller components and solve each of them separately, and the hybrid techniques that combine the other two approaches. In this section, we review the major subcategories of the former group.

Although we could survey numerous large-scale optimizers here, we need to carefully choose a subset of them to keep the size of this section manageable. As a result, we mainly focus on the specific approaches that these algorithms take to address the curse of dimensionality (*e.g.,* improving population initialization or using an adaptive parameter control schema) rather than the type of the optimization technique they use. More precisely, we are not interested in categorizing algorithms to groups such as PSO or DE-based methods. The more important aspect to be reviewed here is how they manage to solve a large-scale problem when the computation budget is limited.

2.4.1 Population Initialization

It is widely believed that beginning the search process from a good starting point will enhance the convergence speed and improve the quality of the final solution using limited resources [Gutiérrez et al. 2011]. Given the simplicity of adopting these techniques, it is reasonable to follow this direction and study the potentials that the advanced population initialization techniques can bring into existence. These improvements are more valuable

when the search space is vast, and the computational budget is restricted. In Chapter 3, we review the initialization techniques in details.

Besides various approaches that were proposed to generate the initial population, some other interesting researches investigated initial population from a different angle. The idea of using tiny initial populations is one of these directions. In [Dasgupta et al. 2009b], for example, a special bacterial foraging algorithm (micro-BFA) was introduced to tackle high-dimensional problems. Similar ideas have been adopted in micro-DE [Olguin-Carbajal et al. 2013b, Salehinejad et al. 2014] and micro-bat algorithm [Topal et al. 2015]. Controversially, some researchers proposed to start the algorithms with large population size and then reduce the size of the population gradually during the search process. Brest *et al.* explored this idea in several studies such as [Brest et al. 2008] and [Brest and Maučec 2011].

Another line of research on population initialization is the notion of *reinitialization*. Reinitializing the population can be beneficial in algorithms that are known for fast convergence. The idea of partial and complete population reinitialization was explored in brainstorm optimization (BSO) [Cheng et al. 2014] and PSO [Budhraja et al. 2013, Cheng et al. 2012] to improve the population diversity and decrease the chance of premature convergence.

Regarding the role of population initialization step in wasting or saving a considerable portion of the computational budget, we devoted the first major part of the dissertation to this topic. Through Chapters 3 to 6, we review the available initialization techniques and study their impact on the performance of optimizers especially when dealing with large-scale problems. Finally, we draw some important conclusions on the proper usage of these techniques when the computational budget is limited.

2.4.2 Advanced Sampling Techniques

One of the main factors that differentiate metaheuristic optimizers from each other is the way they sample search space. Indeed, population initialization (see Section 2.4.1), evolutionary operators, and local searches (see Section 2.4.3) all can be seen as sampling techniques. In this section, we briefly review some generic and scalable sampling techniques that have been used in the optimization process and successfully improved the performance of optimization algorithms. To make the length of our list manageable, we omit the application-specific sampling techniques that can be only adopted in limited algorithms or the techniques that their scalability has not been examined.

Orthogonal Operators

One of the ideas that is widely used in generating new candidate solutions is the notion of *orthogonal operators* which is brought from orthogonal experiment design in statistics. When the set of possible combinations of parameter values for an experiment is too large, it is practically infeasible to assess all combinations. Instead, practitioners follow experimental design methods to select a subset of the possible settings which preserve the maximum diversity. The orthogonal design is one of these methods. Orthogonal operators such as orthogonal population generators or crossovers have been shown to able to produce populations with better diversity [Leung and Wang 2001]. For example, the orthogonal crossover that is adopted in Orthogonal Crossover Differential Evolution (OXDE) enhanced the performance of DE on problems up to 200 dimensions [Wang et al. 2012b]. However, since invoking this crossover is computationally expensive, their application is limited.

Selection Operators

As mentioned, many different ideas in generating new solution sets have been explored in the literature. However, a selection mechanism that can significantly affect the quality of final solutions is still one of the less studied areas in the large-scale optimization. In [Cheng and Jin 2015], a novel selection operator was introduced into PSO which preserves about 50% of the computational budget. At each cycle of the so-called Competitive PSO (CPSO), particles are randomly selected from the current population. The selected particles then compete in a pair-wise competition, and winners move forward into the next generation. The location and velocity of the losers are also updated based on the position and velocity of the winners. Reportedly, CPSO achieved promising results on large-scale problems up to $5,000$ dimensions.

A modified CPSO (MCSO) was proposed in [Mohapatra et al. 2017] where updates two-thirds of the population swarms according to a *tri-competitive* criterion. As a result of this minor modification, MCSO can make a huge difference in the solution quality as it preserves a more significant portion of the computational budget in each iteration. Mohapatra *et al.* claim that their idea improves both the exploration and convergence rates.

The JADE Family

JADE is one the most popular variants of DE that has been successfully applied to high-dimensional problems [Zhang and Sanderson 2009]. The mutation operator that has been adopted in JADE utilizes the information of the top $p\%$ solutions found so far to maintain a better diversity in the population. Besides, an external archive is used to store the records of successful and failed operations. In [Khanum and Jan 2011], a centroid-based initialization is plugged into canonical JADE to improve its performance even further. Since its inception, JADE has been enhanced in different ways. For example, Peng *et al.* introduced a multi-start variation of JADE [Peng et al. 2009].

Opposition-Based Algorithms

Rahnamayan *et al.* were the first researchers to introduce the notion of opposition-based learning into the optimization domain [Rahnamayan et al. 2006; 2007b]. They initially proposed a variant of DE empowered by opposite population initialization and sampling (*a.k.a.* generation jumping) to solve large-scale problems [Rahnamayan et al. 2007a, Rahnamayan and Wang 2008a;b]. The main idea behind the opposition sampling techniques is to consider both the candidate solutions generated by any arbitrary procedure and their opposites concurrently. Indeed, in all generations that the opposition operators activated (*i.e.,* the jumped generations), the original population and the population of the opposite solutions are merged, and then the fittest subset is selected to move forward to the next generation. It has been shown that the chance of finding the global optimum increases when the algorithms consider both original and opposite populations.

The generality of opposite sampling allows the practitioners to easily plug them into various continuous optimizers such as GA [Lin and Wang 2010, Iqbal et al. 2010], PSO [Jabeen et al. 2009, Dong et al. 2012], ABC [El-Abd 2011; 2012], the Harmony Search [Chatterjee et al. 2012], and Bacterial Foraging Optimization algorithm [Mai and Li 2011]. In addition, the simplicity of this idea helped researchers to develop different variations of opposition-based optimizers. Quasi-Oppositional [Rahnamayan et al. 2007c], Quasi-Reflection Opposition [Ergezer et al. 2009], Stochastic Opposition [Park and Lee

2016], and Generalized Opposition [Yang et al. 2011, Wang et al. 2013] operators are just a few examples to name.

In general, these sampling techniques are economic ways of using the limited computational budget, especially when dealing with high-dimensional problems.

2.4.3 Local Searches and Memetic Algorithms

Memetic algorithms and other metaheuristics that leverage the power of local searches have gained popularity in large-scale optimization as some of them achieved the best rank in the past IEEE CEC competition series on large-scale optimization [Molina et al. 2019]. The idea of searching the neighborhood area of the current candidate solutions has been studied for a long time in literature [Fogel et al. 1966]. However, Yang *et al.* are probably the first researchers who borrowed this profound notion into large-scale optimization. Their algorithm which is essentially the combination of neighborhood search and DE (called NSDE) improved the scalability of conventional DE on problems with up to 200 dimensions [Yang et al. 2007b]. Later, they developed self-adaptive NSDE called SaNSDE which borrows the idea of parameter adaptation from self-adaptive DE (*a.k.a.* SaDE [Qin and Suganthan 2005]) to dynamically adapts DE's parameter values [Yang et al. 2008c]. They also showed that SaNSDE performs better than its predecessors on problems with up to $1,000$ dimensions [Yang et al. 2008a]. Since then, SaNSDE has been adopted as the subproblem solver in many decomposition-based algorithms. Because of its popularity, we also use this SaNSDE in some parts of our empirical studies in the following chapters.

Using more than one local search operator is common. For example, Multiple Trajectory Search (MTS) adaptively selects one method among three different local search methods [Tseng and Chen 2008]. Before performing an extensive local search, MTS examines these methods and chooses the one which achieved the best results. Later, Zhao *et al.* expanded MTS by hybridizing it with SaDE [Zhao et al. 2011]. The resulting algorithm has shown superior performance over its predecessors.

A PSO with a dynamic neighborhood search which is based on variable trust region methods was proposed in [Fan et al. 2014]. In their paper, Fan *et al.* claimed that the dynamic neighborhood topology assists particles to better cooperate with their neighbor particles. In turn, this decreases the odds of premature convergence by increasing the chance of exploring useful spaces.

Recently, a Memetic Algorithm (MA) with Constrained Local Search (MACLS) is proposed to reduce the effect of the curse of dimensionality on the conventional MAs [Mehta 2017]. It is claimed that the restrictions that MACLS imposes on the local search operators enhance the optimization capability of the MA. Recently, it is has been shown that adopting semi-parameter adaptation technique in MA framework can significantly improve the effectiveness of MACLS [Hadi et al. 2019].

The idea of local search *chains* that was introduced by Molina *et al.* to the context of MA aims to perform an intensive local search during optimization [Molina et al. 2009]. The term *chains* refers to the fact that a local search operator such as Solis-Wet (SW) random search techniques can be applied sequentially [Solis and Wets 1981]. Besides, each invocation of a local search algorithm can resume the search process from where it stopped previously. Therefore, the whole process forms a chain of local searches that can better exploit the properties of the landscape by focusing on the more promising regions. MA-SW-Chains and MA-SSW-Chains are two of the most popular local search chains algorithms that achieved promising outcomes in solving large-scale problems [Molina et al. 2010; 2011]. The MA-SW-Chains is the winner of the CEC'10 Competition on Large-Scale

Optimization. In the following chapters, we use MA-SW-Chains as one of the state-of-the-art algorithms in the comparative studies.

2.4.4 Adaptive Techniques

The dynamics in the search process of iterative algorithms such as most of the meta-heuristic optimizers demands to adopt some flexibility in the algorithm design and structure to adapt to the changes. For example, to preserve an effective balance between exploration and exploitation, algorithms need to dynamically change the key parameters (*e.g.,* mutation rate in GA or inertia weight in PSO) based the current or even the expected future status of the population. To address this demand, many adaptive techniques have been introduced into optimization. Since the computational budget is even more critical in large-scale optimization, these techniques have been extensively used in different situations, such as adaptive parameter tuning, operator selection, and budget allocation. In the followings, we briefly review some of them.

Hsieh *et al.* designed a PSO algorithm with an Efficient Population Utilization Strategy (*a.k.a.* EPUS-PSO) that adaptively manages the population size [Hsieh et al. 2008]. Generally speaking, if the global best of the swarm is not updated for a long time, the size of the population is increased by generating new particles using a crossover-like operator on the past best solutions. In contrast, when the information content of the swarm is rich enough, some of the poor quality solutions are removed from the swarm. Another approach to adaptively manage population size was proposed by Brest *et al.* for DE [Brest et al. 2008]. This adaptive DE also adopts a sign alteration of scaling factor F (a key DE parameter). The sign of F is exchanged with a probability based on the fitness values of randomly selected vectors (for the mutation) during the search process.

In another attempt, Garcia *et al.* [García-Nieto and Alba 2011] proposed a PSO algorithm with velocity modulation and restarting strategies. The former approach was used to control the movement of particles to be directed within a limited area. The later, however, was employed to prevent premature convergence. Ali proposed a soft adaptive PSO which equipped with adaptive inertia weight mechanism. To maintain exploration-exploitation balance and to improve the performance of the algorithm, the author also recommended an acceleration feature to update the position rule at the next time [Ali 2010].

A DE algorithm based on landscape modality detection (*a.k.a.* Landscape Modality Detection and Diversity archive or LMDEa) was proposed in [Takahama and Sakai 2012]. The modality of the landscape is determined by the number of detected valleys in an arbitrary line; if there is only one valley then the landscape is assumed to be unimodal; otherwise it should be multimodal. The value of scaling factor F then is selected based on the modality of the landscape. In particular, LMDEa uses larger F values for multimodal problems and smaller F values for unimodal functions. It has been shown that LMDEa performs better than DECC-G [Yang et al. 2008a] and MLCC [Yang et al. 2008b] on large-scale problems.

The JADE, which we previously reviewed in Subsection 2.4.2, is also known as an adaptive DE because of its external archive. Another scalable adaptive DE that has been extensively used in large-scale optimization is SaDE [Qin and Suganthan 2005]. In SaDE the DE parameters are tuned based on sampling from various probability distributions. Later, Yang *et al.* have added neighborhood search into SaDE to propose SaNSDE [Yang et al. 2008c]. In another attempt, they combined the adaptation techniques of these three algorithms to design a Generalized DE (GaDE) [Yang et al. 2011]. Other adaptive DEs

that have been used as high-dimensional optimizers are dynNP-DE [Brest and Maučec 2008], jDEdynNP-F [Brest et al. 2008], and jDE [Zhang and Sanderson 2009].

Another exciting innovation in adaptive DE belongs to Shuffle or Update Parallel DE (SOUPDE) which uses a multi-population strategy that can concurrently search different parts of the search space [Weber et al. 2011]. This technique randomly rearranges the individuals across the sub-populations to preserve the population diversity. SOUPDE also dynamically updates F of each sub-population throughout optimization. Some practical experiments have shown the superiority of SOUPDE over older techniques such as CHC [Eshelman 1991] and restarting CMA-ES (G-CMA-ES) [Auger and Hansen 2005] when the dimensionality of the problems was increased to $1,000$.

The idea behind Dynamic Multi-Swarm PSO (DMS-PSO) that proposed in [Liang and Suganthan 2005] is to some extent similar to the multi-population strategy of SOUPDE. In DMS-PSO the neighborhood topology of the particles changes over time according to the perceived from the search dynamics. Later, Zhao *et al.* incorporated a local search operator into DMS-PSO to improve its scalability [Zhao et al. 2008].

The application of adaptive operators and strategies are not limited to DE or PSO. For example, Rashtchi *et al.* proposed an Adaptive Step length Bacterial Foraging algorithm (ASBF) for large-scale optimization to decrease the chance of trapping in local optima [Rashtchi et al. 2009]. As another example, a variant of CMA-ES was proposed that utilizes the search equation and better guide the exploitation process in higher dimensions using the information encoded in the best solution found so far [Fuxing et al. 2017].

Recently, a parameter-free adaptive pattern search for large-scale problems has been proposed that has no parameter visible to users [Gardeux et al. 2017]. It has been claimed the default settings are determined without a priori experimentation and the algorithm's performance is highly robust on the benchmarked problems.

2.4.5 Function Approximation and Surrogate Models

The use of function approximation models in metaheuristic algorithms has received increasing attention because many real-world optimization applications are computationally expensive [Jin 2011]. These techniques which are also called *surrogate assisted* optimizers usually use an approximation of the actual search landscape to call the expensive objective function less often [Shi and Rasheed 2010]. As a result, they can save a considerably large portion of the computational budget with the expense of introducing some uncertainty into the decision as the approximations are rarely error-free [Jin 2005].

One of the main challenges in adopting surrogate assisted metaheuristics is their scalability [Sun et al. 2016]. To have a reliable fitness approximation or at least precise solution ranking, we usually need adequate training samples. When it comes to high-dimensional problems, the size of training dataset should be even more significant (usually with an exponential rate) which is impractical in many real-world applications.

We can categorize the surrogate models to two large families of general and local models. Due to the sparsity of training set in large-scale optimization, most of the general models that try to approximate the original fitness function become inaccurate. This lack of accuracy may introduce false optimums and mislead the search process [Jin 2011]. In contrast, local surrogate models which aim to approximate a bounded region of the search space may achieve better results because accurately modeling these small areas of the landscape is relatively more straightforward to make even with a smaller training set.

The idea of Baldwinian trust-region that was adopted in [Ong et al. 2006] is an example of local surrogate models. The method has been shown to be successful in

reducing the computational cost when examined on synthetic and real-world problems *e.g.,* aerodynamic shape design tasks. In another attempt, Le *et al.* proposed an adaptive selection strategy to chose one model among multiple surrogate models for each single individual [Le et al. 2013].

The idea of using both global and local models is also prevalent. For example, Zhou *et al.* employed a global surrogate to preselect promising individuals, and then, a trust-region method was used to perform a local search on a local surrogate model [Zhou et al. 2007]. In several other work such as MA in [Lim et al. 2010] and PSO in [Sun et al. 2015a], global and local models were employed together, either simultaneously or sequentially. In these algorithms, a global model is usually used to smooth the rugged fitness landscape while the local models provide more accurate approximations in limited areas.

On a different line of research, Sun *et al.* proposed a fitness approximation assisted competitive PSO to estimate the fitness according to the spatial relationship between individuals instead of approximating the fitness landscape itself which is prone to the curse of dimensionality [Sun et al. 2016]. The published empirical studies on problems up to 500 dimensions show that the proposed surrogate assisted PSO achieved competitive results using a limited computational budget.

In another recent attempt, Wang *et al.* borrowed the idea of committee-based decision making from active learning domain into PSO [Wang et al. 2017]. They suggested to find the best and most uncertain solutions according to the surrogate ensemble and evaluates them using the original objective function. Then, they built a local surrogate model around the current best solution and employed a PSO algorithm to search the surrounding area. Once no further improvement can be observed they switch from global model management strategy to the local search, and vice versa.

2.5 Decomposition-Based Techniques

The adoption of divide-and-conquer strategy is very common when solving large-scale optimization problems [Yazdani et al. 2019]. A decomposition-based algorithm tries to divide a high-dimensional objective function into a set of smaller and simpler subproblems (*a.k.a.* components) and optimize each of them separately. Generally, this strategy includes two tasks:

1. decomposition of the original problem into the set of subproblems, and
2. optimization of subproblems either concurrently or sequentially.

The first task is generally governed by the underlying interaction structure of the decision variables. The level of interactions between variables strongly affects the complexity of the decomposition process and the effectiveness of the technique. The order of solving the subproblems and amount of budget to be allocated to each of them can also significantly influence the outcome. Traditionally, the subproblems are solved in a round-robin fashion while each of them is granted the same portion of computational resources regardless of their dimension size, search landscape features, and complexity. Before reviewing the decomposition-based optimization techniques, we first need to formally define some fundamental concepts such as variable interaction, separable and nonseparable problems.

2.5.1 Variable Interaction

The term *epistasis*, originaled from natural genetics, refers to any type of gene interaction [Davidor, Klug et al. 2008]. For example, two or more genes are called interacting

if they collectively represent one phenotype feature, or specific values of one of them activates or deactivates the effect of the other gene(s) [Ptashne 1989]. In the optimization literature, the terms *linkage* and *nonseparability* are used to refer to similar concept [Chen et al. 2007, Weise et al. 2012]. In the followings, we formally define these terms and related concepts.

Definition 5 (Variable Interaction)**.** A variable x_i is separable from the rest of variables iff:

$$\arg\min_{\mathbf{x}} f(\mathbf{x}) = \Big(\arg\min_{x_i} f(\mathbf{x}), \arg\min_{\forall x_j, j\neq i} f(\mathbf{x})\Big),$$

where $\mathbf{x} \in \mathbb{R}^D$, and D is the number of variables. Here, the notation $\overset{*}{\mathbf{x}}_i = \arg\min_{x_i} f(\mathbf{x})$ indicates the optimum value of x_i according to f when all other variables are kept constant.

Definition 6 (Separable Function)**.** An objective function $f(\mathbf{x})$ is separable iff:

$$\arg\min_{\mathbf{x}} f(\mathbf{x}) = \Big(\arg\min_{\mathbf{x}_1} f(\mathbf{x}_1, \dots), \dots, \arg\min_{\mathbf{x}_K} f(\dots, \mathbf{x}_K)\Big),$$

where for all $k \in \{1, \dots, K\}$ the $\mathbf{x}_k$ are disjoint subvectors (components) of $\mathbf{x}$ and $K \in \{2, \dots, D\}$ is the number of subproblems. Let $\mathcal{D}_k$ be a set that denotes the indices of the variables from $\mathbf{x}$ which are also included in $\mathbf{x}_k$. Then, the notation $\arg\min_{\mathbf{x}_k} f(\dots, \mathbf{x}_k, \dots)$ indicates the optimum value of f when the values of all variables that their indices are not in $\mathcal{D}_k$ are fixed. For simplicity, we may refer to $f(\dots, \mathbf{x}_k, \dots)$ as $f_k(\mathbf{x})$ or $f_k(\mathbf{x}_k)$, and to $\arg\min_{\mathbf{x}_k} f(\dots, \mathbf{x}_k, \dots)$ as $\overset{*}{\mathbf{x}}_k$. As a result, $\overset{*}{\mathbf{x}} = \Big(\overset{*}{\mathbf{x}}_1, \dots, \overset{*}{\mathbf{x}}_K\Big)$.

Definition 7 (Fully Separable Function)**.** A separable function f is fully separable iff in Definition 6, $D = K$. As a result[3], $|\mathbf{x}_k| = |\mathcal{D}_k| = 1$ for all $k \in \{1, \dots, K\}$. This means all variables are separable from other variables or $\overset{*}{\mathbf{x}} = \Big(\overset{*}{x}_1, \dots, \overset{*}{x}_D\Big)$.

Definition 8 (Partially Separable Function)**.** A separable function f is partially separable iff it is not a fully separable function. In other words, there is at least one k such that $|\mathbf{x}_k| = |\mathcal{D}_k| > 1$. As a result, $1 < K < D$.

Definition 9 (Partially Additively Separable Function)**.** A separable function f is partially additively separable if it can be written in the following form:

$$f(\mathbf{x}) = \sum_{k=1}^{K} C_k \cdot f_k(\mathbf{x}_k),$$

where $C_k \in \mathbb{R}$ are arbitrary constant coefficients.

Definition 10 (Nonseparable Function)**.** Trivially, an objective function f is nonseparable if it does not satisfy the separability conditions in Definition 6.

Definition 11 (Decomposition)**.** A decomposition $\mathcal{D}$ is a set of sets such that $\mathcal{D} = \{\mathcal{D}_1, \dots, \mathcal{D}_K\}$ where $\mathcal{D}_k \subseteq \{1, \dots, D\}$ and $\bigcup_{k=1}^{K} \mathcal{D}_k = \{1, \dots, D\}$. As a result, $\mathbf{x}_k = \{x_i\}$ where $i \in \mathcal{D}_k$. In general, a component $\mathcal{D}_k$ may have overlap with one or more components. More formally, $|\mathcal{D}_k \cap \mathcal{D}_l| \geq 0$.

[3] In this context, the cardinality function $|.|$ returns the length of a vector as in $|\mathbf{x}|$ or the size of a set as in $|\mathcal{D}|$

Definition 12 (General Ideal Decompositions)**.** An ideal decomposition $\mathcal{D}$ is a decomposition where there is no overlap between any two components (*i.e.*, $\forall k, \forall l$ where $k \neq l$, then $\mathcal{D}_k \cap \mathcal{D}_l = \varnothing$) and no variable from one component should have any interaction to another variable from any other component. More formally, $\forall k, \forall l$ where $k \neq l$ and $\forall i \in \mathcal{D}_k$, then x_i is separable from $f_l(\mathbf{x}_l)$ according to Definition 5).

Based on Definition 12, solving each of the subproblems separately while other subproblems are kept fixed should result in solving the original function f. Note that there can be more than one general ideal decomposition for a given function. For example, any decomposition of a fully separable function with non-overlapping components is a general ideal decomposition. For example, the number of general ideal decompositions for fully separable functions of size 10, 100, and 1000 variables are larger than 10^5, 10^{115}, and 10^{1927}, respectively. The exact number of such partitions can be calculated using Bell's recursive summation formula [Gardner 1978].

In the EC literature, the term ideal decomposition usually refers to a unique type of general ideal decomposition which can be defined as follows.

Definition 13 (The Ideal Decomposition)**.** A general ideal decomposition is referred to as the ideal decomposition iff none of its components can be decomposed further into smaller separable functions. In other words, it is a unique general ideal decomposition with the maximum number of separable components.

Based on Definition 13, the only ideal decomposition for a fully separable function is the one that each component consists of exactly one variable. Note that as examined in [Omidvar et al. 2014b], the ideal decomposition is not necessarily the most effective decomposition in practice.

2.5.2 Decomposition Methods

The famous divide-and-conquer strategy is a generic problem-solving approach that has been practised for solving complex problems. Decomposing a complex objective function into smaller and hopefully simpler subproblems is indeed a relatively old strategy [Dantzig and Wolfe 1960]. In the early '70s, for example, the structural information of large-scale linear programming problems was used to break problems into smaller subproblems [Dantzig and Wolfe 1960, Benders 1962]. As another example, a quasi-Newton method was used to tackle high-dimensional partially separable optimization problems [Griewank and Toint 1982]. Similar ideas have been employed into metaheuristics to lessen the effect of dimensionality.

A problem may be decomposed manually by exploiting expert knowledge or automatically when a decomposition algorithm is adopted. A manual decomposition can be useful only when adequate domain knowledge exists. In black-box optimization problems, however, such comprehensive and accurate knowledge-base is not available. Therefore, optimal manual decomposition is practically unattainable.

The automatic decomposition techniques can be divided further into implicit and explicit grouping algorithms. Estimation of Distribution Algorithms (EDAs) are the best examples of former category as they implicitly perform the decomposition by learning a collection of probability distributions each approximates a group of interactive variables [Mühlenbein and Paass 1996]. On the other hand, many specific decomposition techniques have been proposed to incorporate with Cooperative Coevolution algorithms (CC) [Potter and De Jong 1994].

Decomposition algorithms can also be categorized into static (off-line) and dynamic (online) techniques. The former refers to methods which group the variables only once at the start of the optimization process and then never alter these components. In contrast, dynamic techniques revise the components many times during the optimization process. In general, it is practically easier to monitor the performance of static groups or manage their budgeting in comparison with the dynamic components that may emerge or become depreciated anytime in the course of optimization. However, the dynamic optimization better handles the movements of the population from one state to another. For example, a dynamic decomposition may merge a few small subproblems to decrease the overhead or subdivide a complex subproblem into further components which might be easier to solve separately.

Omidvar categorizes the decomposition techniques into four groups, namely: random methods, perturbation techniques, interaction adaptation methods, and probabilistic model learning algorithms [Omidvar 2015]. In the following, we briefly review some examples of these categories.

Random Variable Grouping

Random variables grouping is the simplest way to subdivide a problem into smaller pieces. These methods do not make any effort to discover the interaction between variables. Instead, they randomly select a subset of variables to form the components. A dynamic random grouping may repermute the decision variables several times during the optimization to increase the probability of placing interacting variables close to each other for a few evolutionary cycles. The grouping can be done only once at the beginning of the algorithm (*a.k.a.* fixed or static random grouping) or repeated every few iterations (*a.k.a.* dynamic random grouping). The inversion operator is probably the first attempt to address the variable linkage problem by reversing the order of variables in a randomly chosen part of a candidate solution [Holland 1975, Goldberg et al. 1989].

On the other hand, the so-called CCGA-1 is the first attempt to implement a Cooperative Coevolutionary GA [Potter and De Jong 1994] which adopts a static random grouping approach. In their paper, Potter and De Jong proposed to divide a D-dimensional problem into D 1-dimensional subproblems and solve each of them using a variant of GA in a round-robin fashion. Liu *et al.* borrowed this idea and developed a Fast Evolutionary Programming in the context of the CC framework to tackle large-scale problems [Liu et al. 2001]. The experimental results on functions with up to $1,000$ dimensions showed that the relative scalability of a CC model improves as the dimensionality of the problem increases. However, since the applied decomposition assumes the given problem is fully separable, the CC algorithms did not perform well on nonseparable problems *e.g.,* Griewank and Rosenbrock functions [Tang et al. 2009].

There are two issues associated with this simple decomposition strategy. Firstly, it assumes that the problem is fully separable which is not the case in many real-world applications. As mentioned earlier, ignoring the linkage between variables and grouping correlated variables into different groups may significantly reduce the performance of optimizer. Secondly, having D subproblem imposes a great deal of overhead onto the CC framework. Recently, Omidvar *et al.* showed that creating larger groups even when the smaller separable components are available may significantly improve the performance of a CC algorithm [Omidvar et al. 2014b].

Ten years after the introduction of CCGA-1, van den Bergh and Engelbrecht developed the first CC PSO (CPSO) but with a different decomposition strategy. They decomposed a D-dimensional problem into K S-dimensional subproblems where $S < D$ [van den

Bergh and Engelbrecht 2004]. This strategy has three limitations. Firstly, the user should find the optimum value for K or S, which is impractical in black-box problems. Secondly, in the case of partially separable problems, the size of subproblem may not be equal. Therefore, even the best value of S may not fit the size of many of subproblems. Thirdly, nor the user neither the algorithm have any means to determine the variable linkage to group the correlated variables of a black-box problem into the same partition. The decomposition that CPSO adopted can outperform CCGA-1 in fully separable and fully nonseparable problems only if the parameters are chosen correctly. In the case of partially separable functions, both decomposition strategies are inefficient.

The term *random grouping* gained its popularity in optimization from an early work of Yang *et al.* who proposed to randomly permute the order of the decision variables in every cycle of a CC to increase the probability of placing two interacting variables in the same group for at least one cycle [Yang et al. 2008a]. A cycle in the CC framework is defined as a series of consecutive subproblem optimization attempts such that each subproblem is selected exactly once. A selected subproblem may be optimized for one or more evolutionary iterations which called one epoch. It is generally advised to increase the frequency of permutation by decreasing the number of iterations in each epoch to lessen the adverse effect of dimensionality [Omidvar et al. 2010a]. More frequent permutation increases the chance of grouping interacting variables into one partition for at least one epoch.

Since its invention, the notion of random grouping has been used in many variants of large-scale optimizers such as CCPSO [Li and Yao 2009], CCPSO2 [Li and Yao 2012], and CCOABC [Ren and Wu 2013]. However, a significant limitation of such random groupings is the fact the practitioner must choose the size of each subproblem which may not necessarily be an optimal decision. However, this is not the biggest problem. The main issue with the blind decompositions is that the odds of having all interacting variables in one group, even for a single cycle, approaches to zero as the dimensionality of the problem increases [Omidvar et al. 2010b].

Perturbation-based Techniques

These mostly static decomposition techniques randomly group the variables in the initial stage. Then, they observe the groups' performance by optimizing a few samples from each group. They may perturb the variables several times according to the perceived feedbacks from objective function to improve the decomposition performance. Depending on the adopted heuristics, perturbation techniques may consume a great deal of the computational budget upfront, rather than investing it in the actual optimization phase.

There are many instances of perturbation algorithms that employed independently or as a part of CC framework. The Fast Messy GA (fmGA) [Goldberg et al. 1993], Gene Expression Messy (gemGA) [Kargupta 1996], GA with Linkage Identification by Nonlinearity Check (LINC) [Munetomo and Goldberg 1999] and its real-value extension (LINC-R) [Tezuka et al. 2004] are some famous examples of standalone GA-based perturbation techniques that mostly used for tackling binary problems.

In the context of large-scale optimization, many instances of perturbation-based decomposition techniques have been developed. Coevolutionary optimization [Weicker and Weicker 1999] and CC with Variable Interaction Learning (CCVIL) [Chen et al. 2011a] are just two famous examples to name. These decomposition techniques follow different approaches to find the linkage between variables. For example, Ray *et al.* proposed a CC Algorithm using Correlation-based Adaptive variable Partitioning (CCEA-AVP) that uses Pearson correlation coefficient as the variable interaction indicator. They optimized

the whole problem for five consecutive iterations, identified the top 50% of the population and then calculate the correlation between variables using these points. Finally, the highly correlated variables are grouped into one partition and all other variables in another partition. Singh *et al.* improved this technique and proposed AVP2 that was able to create variable partitions with varying sizes [Singh and Ray 2010]. Another correlation-based decomposition called 4CDE was proposed by Rojas *et al.* [Rojas and Landa 2011]. The main difference between 4CDE and AVP2 is that the former calculated the correlation between variables and the objective function, while the later partitions the space based on the relationship between all pairs of variables.

It has been argued that the improvement interval of interacting variables is relatively smaller than those of noninteracting variables [Salomon 1995]. Based on that, Omidvar *et al.* proposed a non-correlational approach for decomposition called Delta Grouping [Omidvar et al. 2010b]. This method calculates the dimension-wise displacement of the candidate solutions over the entire population between two consecutive runs and then sorts the decision variables accordingly. The Delta Grouping groups variables into K partitions of size S based on the calculated values.

Later, Omidvar *et al.* proposed the famous Differential Grouping (DG) technique based on a simple fact that is directly derived from the definition of partially separable problems (see Definition 8) [Omidvar et al. 2014a]. DG groups the variables based on the fact that two arbitrary variables are labeled as interacting variables if the forward difference of one of them is significantly shifted when the value of the other variable changes. Otherwise, DG assumes the two variables are separable. In a more recent work, a faster and more accurate differential grouping called DG2 is proposed [Omidvar et al. 2017]. It has been shown that DG2 is more robust in dealing with functions having overlapping components. It also eliminates the ϵ value which was previously used as a user-defined threshold to determine whether the forward difference is shifted significantly. The Global Differential Grouping (GDG) and eXtended Differential Grouping (XDG) are other variants of DG that are proposed to enhance the sensitivity of DG to ϵ value and improve its accuracy in dealing with overlapping functions [Mei et al. 2016, Sun et al. 2015b].

In a separate line of research, Mahdavi *et al.* employed the High Dimensional Model Representation (HDMR) as a mean for problem decomposition [Mahdavi et al. 2014a]. The HDMR method defines a set of representative models which can capture high-dimensional input-output system behavior [Sobol 2003]. These models are used in discovering the unknown arrangement of components in a black-box problem.

Interaction Adaptation Techniques

As opposed to perturbation-based techniques, these dynamic decomposition methods evolve the decomposition structure during the search process. As the term adaptation implies, interaction adaptation techniques adopt a detection mechanism to find the linkage between variables. At the same time, they evolve the order of the decision variables of the original optimization problem. Finally, they assign a higher reproduction chance to the candidate solutions with a tighter grouping of interacting variables. The Linkage Evolving Genetic Operator (LEGO) [Smith and Fogarty 1995] and Learning Gene Linkage GA (LGLGA) [Harik 1997] are two representatives of this category.

Probabilistic Model Learning Techniques

Most of the EDAs fall into this category. These techniques implicitly learn one or more models based on promising candidate solutions in the population and continuously update the models as the population evolves [Pelikan et al. 2002]. Therefore, the new generation of candidate solutions is generated based on the last state of the model such that the chance of producing new solutions around the current good candidates are higher than sampling any other part of the landscape. The Compact GA (cGA) [Harik et al. 1999], Iterated Density Estimation Algorithm (IDEA) [Bosman and Thierens 2000], Bayesian Optimization Algorithm [Pelikan and Goldberg 1999], Hierarchical BOA (HBOA) [Pelikan et al. 2001], and Probabilistic Model-Building GA (PMBGA) [Pelikan 2011] are some of the widespread instances of model learning techniques.

As the dimensionality of problems grows, the sufficient sample size for typical EDAs exponentially increases [Friedman et al. 2001]. This vital demand hinders the scalability of EDAs as a small population size dramatically degrades their performance [Vershynin 2010]. Even if such huge population size is available, the cost of sampling from a high-dimensional statistical distribution can be an obstacle in the scalability of EDA as it cubically increases with the dimensionality of the problem [Dong and Yao 2008].

Several EDA variants have been developed to lessen the effect of dimensionality. For example, Wang *et al.* combined the Gaussian with Lévy or Cauchy distributions to improve the exploration-exploitation balance in LSEDA-gl and MUEDA, respectively [Wang and Li 2008]. Both of these univariate models perform well on separable functions. However, theoretical and empirical studies have shown that the univariate EDAs cannot capture the complex linkage between variables. Therefore, they are not recommended for solving nonseparable problems [Muhlenbein and Mahnig 1999, Larrañaga and Lozano 2002, Echegoyen et al. 2011].

Another attempt that has been made by Dong *et al.* to improve the scalability of EDAs is Estimation of Distribution Algorithm with Model Complexity Control (EDA-MCC) [Dong et al. 2013]. To identify *weakly dependent variables* the EDA-MCC applies a threshold on the Pearson correlation coefficient matrix of sample points, and then, uses a simple EDA to optimize them. Besides, it randomly projects the current candidate solutions down into several subspaces of the original space. This helps EDA-MCC to perform *subspace modeling* and avoid large sample sizes for generating an accurate model.

In a separate line of research, Kabán *et al.* used an ensemble of projected points and then estimate the covariance of the population in each subspace [Kabán et al. 2016]. Then a new population in the original space is constructed using these ensembles. They showed that a proper combination of the ensembles would result in a smoothing effect that advances the exploration capability of EDA. Later, Sanyang *et al.* have shown that a multivariate Cauchy is advantageous over Gaussian distributions [Sanyang and Kaban 2014]. In a more recent study, they took advantage of the degrees of freedom of a t-distribution as a tunable parameter to control the exploration-exploitation balance of their EDA [Sanyang and Kabán 2015].

The Ensemble Optimization EA (EOEA) is another scalable EDA that consists of two building blocks: (1) a global shrinking stage which can be performed by any arbitrary variant of EDA and (2) a local exploration stage which is enforced using a CC algorithm [Wang et al. 2013].

The notion of random projection shown to be very useful in the decomposing original large-scale problem into an ensemble of low-dimensional subproblems. Theoretically, it is possible to preserve important features of the original space, *e.g.,* Euclidean distances or dot products, in the new space within a reasonable tolerance [Dasgupta 1999]. Further-

more, the projection has some desirable side effects such as the distribution of individuals generally become more Gaussian in the reduced space [Diaconis and Freedman, 1984]. These features make it possible to capture the variables linkage while consuming less computational resources. In other words, we can avoid oversampling (as in traditional EDAs) and oversimplification of the models (as in the case of univariate models). The downside of this approach is that a random projection of solutions into a subspace might not be the best decomposition strategy for solving partially separable problems.

2.5.3 Curse of Imbalanced Contribution

Most of the decomposition-based metaheuristics, especially the CC models, assume that all subproblems are equally important and solving each of them is equally challenging. As a result, they allocate the computational budget (regarding population size and the number of iterations) uniformly among all subproblems. In practice, however, many applications are identified to have components with unequal importance and difficulty [Omidvar et al. 2011]. The source of the imbalance contributions of components in the quality of the final solution can be any combination of nonuniform dimensionality, non-identical search landscape, and unequal coefficients (especially in partially additive separable problems). Therefore, the traditional round-robin CC algorithms may not perform well when such an imbalance exists.

To lessen the effect of imbalance on the performance of CC algorithms, several studies have been published. The first attempt to address this shortcoming is made by Omidvar *et al.* who proposed a simple metric to measure the contribution of each component in improving the current best solution. Then, they developed two heuristics known as Contribution-Based CCs (CBCC1 and CBCC2) to allocate more computational resources (in terms of more optimization epochs) to the most contributing subproblem.

In Chapter 7 we conduct a series of sensitivity analysis on the performance of CBCCs. We show that both algorithms suffer from a poor balance in spending the resources in the exploration and exploitation phases. In the context of CBCC, an exploration phase is the one that the algorithms optimizes all subproblems for one epoch each to measure their contributions. Likewise, the exploitation phase refers to the cycle that only the most contributing component is optimized for one or more epochs. To maintain a higher level of exploration-exploitation balance, they proposed CBCC3. It has been empirically shown that CBCC3 outperforms its predecessors while it is less sensitive to the values of the critical parameters [Omidvar et al. 2016].

Another approach to address the imbalance contribution is to use sensitivity analysis tools [Mahdavi et al. 2016b;c; 2017]. For example, Mahdavi *et al.* proposed Multilevel Optimization Framework Based on Variables Effect (MOFBVE) which employs Morris screening to compute the main effect of variables. The MOFBVE then constructs variable partitions using the K-means clustering algorithm to group variables with similar effects on the fitness value into related clusters. Finally, the clusters are sorted based on their contribution on the fitness value and optimized accordingly.

A very recent attempt to address the resource management in CC when contribution imbalance exists is the so-called new CC framework (CCFR) [Yang et al. 2017; 2019]. The CCFR expands the resource allocation proposed in CBCCs by adding a stagnation detection technique into the framework. Therefore, CCFR not only tries to increase the allocated budget to the most contributive component but also decreases the waste of resources on stagnated components. Due to the importance of this topic, we devote Chapter 7–10 to the curse of imbalanced components and contribution-aware techniques.

2.6 Hybrid Techniques

We can subdivide hybrid large-scale optimization techniques into three classes: 1) hybrid nondecompositional techniques, 2) hybrid decomposition-based techniques, and 3) hybrid heterogeneous techniques that combine decomposition-based algorithms with nondecompositional algorithms. In the following, we briefly review some representatives from each category.

2.6.1 Hybrid Nondecompositional Techniques

Multiple Offspring Sampling (MOS) is the most famous example of hybrid techniques [LaTorre de la Fuente 2009]. The MOS achieved its fame due to winning multiple numerical competitions *e.g.,* CEC'13 and CEC'15 competitions on large-scale global optimization. The MOS framework is responsible for generating a new population utilizing the available reproductive techniques. On the other hand, the hybridized algorithms dynamically evaluate the performance of their reproduction operators and adjust their participation accordingly [LaTorre et al. 2013]

A number of reproduction operators have been employed in the MOS framework. For example, a combination of MTS-LS1 which is local search strategy borrowed from MTS ([Tseng and Chen 2008]) and DE is proposed in [LaTorre et al. 2011]. In a more recent work, a hybridization of MTS-LS1 and Solis-Wets' randomized hill climber ([Solis and Wets 1981]) introduced in [LaTorre et al. 2012]. A more complex MOS is proposed in [LaTorre et al. 2013] which is a mixture of MTS-LS-Reduced (an enhanced variation of MTS-LS1), Solis-Wets' randomized hill climber, and GA with BLX crossover and Gaussian mutation operators [Herrera and Lozano 2000].

In a separate series of studies, Garca`-Martn` *et al.* developed a continuous variable neighborhood search algorithm with three primary building-blocks: generation, improvement, and shaking [García-Martínez and Lozano 2009]. These elements employ Covariance Matrix Adaptation Evolution Strategy ([Hansen and Ostermeier 2001]) as generator component, a continuous local EA ([Herrera and Lozano 2000]) as improvement component and the combination of Micro Cross-generational elitist selection, Heterogeneous recombination, and Cataclysmic mutation ([Lozano and García-Martínez 2008]) as the shaking components.

The extensive use of local search in hybrid techniques is not limited to the algorithms mentioned above. For example, de Oca *et al.* proposed an incremental population size variant of PSO that was hybridized with local search [de Oca et al. 2011]. In a similar study, a new incremental variant of ACO is combined with three local search algorithms (*i.e.,* Conjugate Directions Set [Powell 1964], BOBYQA [Powell 2009], and MTSLS [Tseng and Chen 2008].) and applied to large-scale problems [Liao et al. 2011]. The incremental facility introduced in IACO$_{\mathrm{R}}$-LS further diversifies the constantly growing solution archive and, at the same time, the employed local search tool enhances the search intensification abilities of the algorithm.

It is also common to mix minimalistic variations of metaheuristics to keep the computational budget manageable [Arellano-Verdejo et al. 2014]. For instance, in [Yildiz et al. 2015] a micro-Chemotaxis Differential Evolution Optimization Algorithm (CDEOA) is combined with canonical DE, and a mixture of micro-DE and local search (MTS [Tseng and Chen 2008]) is proposed in [Olguin-Carbajal et al. 2013a]. In another attempt, the foraging mechanism of E-coli bacterium (borrowed from Bacterial Foraging Algorithm [Passino 2002]) with the swarming pattern of birds in a block (as in PSO) are combined to develop a Fast Bacterial Swarming Algorithm (FBSA) [Chu et al. 2008].

2.6.2 Hybrid Decomposition Techniques

The combination of several decomposition-based techniques is not as common as other categories of hybrid techniques. The work of Liu *et al.* is one of the implementations of such hybridization [Liu and Tang 2013]. In their paper, they proposed a CC variant of CMA-ES which in each cycle adaptively selects a decomposition from a pool of three strategies: simple random grouping, Min-Variance Decomposition, and Max-Variance Decomposition, and used one of them at a time.

In a similar approach, Omidvar *et al.* proposed a useful decomposition framework for fully-separable problems that dynamically selects a decomposition schema from a predefined pool of available group sizes and tracks their performance [Omidvar et al. 2014b]. In the next cycle, the so-called MLSoft selects the grouping size with the highest expected reward from the pool and randomly groups the variables accordingly. It has been shown that MLSoft can significantly outperform ideal decomposition as it decreases the overhead of having a large number of unnecessarily small components.

2.6.3 Hybrid Heterogeneous Techniques

One of the common practices in large-scale optimization is to mix a decomposition-based technique (usually a CC model) with other nondecompositional algorithms *e.g.,* alternative population initialization, smart sampling techniques, or adaptive parameter tuning. Indeed, the generality of the CC framework makes it ideal to be incorporated with other advanced optimization techniques. For example, Mahdavi *et al.* introduced a Center-based initialization schema for the CC framework which outperformed its predecessors [Mahdavi et al. 2016a].

Zamuda *et al.* introduced a log-normal self-adaptation technique into the CC framework to adaptively control the DE parameters as the CC's subproblem optimizer [Zamuda et al. 2008]. The so-called DE with Self-Adaptation and CC (or DEwSAcc for short) was shown to outperform similar self-adaptive DE with omitted CC mechanism significantly.

In an attempt to reduce the computational costs of CC performance, Parsopoulos plugged a micro-DE ([Rahnamayan and Tizhoosh 2008]) into CC framework [Parsopoulos 2009]. The resulting algorithm which is called Cooperative Micro-DE (COMDE) also leverages several techniques to avoid diversity loss in the population.

Ge *et al.* developed two variants of adaptive CC algorithms for solving large-scale problems. In the earlier work, they used a circular sliding window as a mean of problem decomposition [Ge et al. 2016]. In their novel strategy, the window size represents the size of the subproblems which can be easily adjusted based on the separability of the problem. Besides, the sliding step size can also be changed according to the activeness of different regions of the problem. Finally, they adopted an adaptive hybrid DE to tune the algorithm's parameters [Ge et al. 2016] adaptively. In a more recent work, they proposed a bi-level decomposition technique called Two Stage Variable Interaction Reconstruction which is incorporated with a cooperative hierarchical PSO algorithm [Ge et al. 2017]. In the first stage, a learning model is adopted to explore the variable interactions partially. Next, a marginalized denoising model is employed to construct the complete variable interactions using the knowledge gained in the first stage. Then, cooperative hierarchical PSO is used to solve the subproblems. The contingency leadership, interactional cognition, and self-directed exploitation are also adopted in the proposed PSO algorithm [Ge et al. 2017].

Fuzzy decomposition technique is another attempt to improve the CC framework by mixing it with other algorithms [Fan et al. 2014]. The algorithm which called is Fuzzy

kernel clustering, and variable Trust region for Dynamic Neighborhood in PSO (FT-DNPSO) adopts a fuzzy C-means clustering algorithm for decomposition purposes. Then, the variable ranges are changed adaptively by means of variable trust region learning technique. Furthermore, the dynamic neighborhood topology is used to avoid premature convergence in the PSO algorithm.

2.7 Chapter Summary

In this chapter, we formally defined the large-scale optimization problem and related concepts such as separability and imbalanced components to set the foundation for this piece of research. Besides, to demonstrate the big picture of the advances in large-scale optimization, we briefly reviewed different ideas that have been studied in this domain. In particular, we looked at the main families of scalable metaheuristics such as advanced initialization techniques, adaptive strategies, decomposition-based frameworks, and hybrid approaches.

In the following chapters, we fill some of the research gaps in the literature that we identified in this chapter. Our review revealed that despite the popularity of advanced population initializers in metaheuristics, no comprehensive taxonomy had been built to categorize and study them systematically. Therefore, we devote Chapter 3 to deeply review the existing initialization techniques, especially those that have been used in high-dimensional settings. In Chapter 4, we investigate the potential benefits of adopting alternative initialization methods in saving a considerable portion of the computational budget when solving large-scale optimization tasks. In our literature review, we also noticed that researchers used many different parameter settings in their work without justifying why such values have been chosen. Therefore, we dedicate Chapter 5 to analyzing the impact of parameter configuration on the effectiveness of advanced initializers in economical use of computational resources. Furthermore, we found that some of the previously published studies on large-scale initializers claim contradicting results. Therefore, in the last chapter of Part I (Chapter 6), we take a more in-depth look into the practical limitations of uniform population initialization algorithms in high-dimensional spaces.

Besides population initialization, we observed that the adaptive resource allocation is also not explored enough in the past researches. In particular, when a large-scale problem is decomposed into heterogeneous components with noticeably different contributions in the quality of the solution of the original problem, adopting a nonuniform budgeting schema can significantly improve the final outcome. Therefore, we devote Part II of this thesis to review contribution-aware cooperative coevolutionary techniques (Chapter 7) and improving these techniques by maintaining a better balance between exploration and exploitation attempts in component space (Chapter 8). Our literature review also unveiled that the popular benchmarking testbeds lack the required features to analyze all aspects of the contribution-aware techniques. Hence, we build a very comprehensive set of test functions in Chapter 9. Finally, in Chapter 10, we propose a generic budget allocation framework for cooperative coevolutionary algorithms based on Reinforcement Learning concepts to take another step towards a smarter use of limited resources when solving large-scale black-box optimization problems.

Part I

Population Initialization

CHAPTER 3

Population Initialization in Higher Dimensions

In this chapter, we review a wide range of population initialization methods that have been used in the metaheuristics. Then, we propose a new categorization schema that looks into the population initialization from three different aspects; randomness, compositionality, and generality. This categorization helps us to study the scalability of the available techniques to higher dimensions. Finally, we provide a guideline on the application of each family of initializers in solving complex large-scale optimization problems.

3.1 Introduction

Despite all the differences and the considerable diversity of metaheuristic models, almost all of them share one common algorithmic step: the *population initialization*. The role of this step is to provide an initial set of candidate solutions. Then, these potential solutions will be iteratively improved in the course of the optimization process until a stopping criterion is met. It is widely believed that a *good* set of starting points can facilitate an optimizer to locate the optima [Clerc 2008] whereas beginning from a *bad* set of candidates may prevent the algorithm from finding the optima [Maaranen et al. 2004] in the given time-frame. The effect of the quality of the initial population on the final outcome can be more crucial when solving large-scale optimization problems using a limited computational budget [Helwig and Wanka 2008].

In black-box optimization problems, which metaheuristics are apt to deal with, there exists no prior information about the search landscape of a given problem [Chou and Chen 2000]. Therefore, *good* and *bad* initial populations cannot be determined before the execution of the optimizer. In such a case, practitioners often employ pseudo-random number generators (PRNGs) to produce the initial population [Kimura and Matsumura 2005]. The rationale behind this common practice is that PRNGs can presumably generate evenly distributed samples [Ma and Vandenbosch 2012], and thus, a population produced using these techniques tends to cover more regions of the search space [Maaranen et al. 2004]. Having such uniformly scattered populations, when applicable, can increase the likelihood and pace of approaching global or satisfactory local optima located in any unknown regions [Chou and Chen 2000, Maaranen et al. 2007].

Since the population size is always limited, the odd for an initial population to sample all promising regions of the search space decreases as the dimensionality of the search space

growth [Helwig and Wanka 2008, Pant et al. 2007]. In other words, the volume of spaces between the individual solutions exponentially increases as the dimensionality of the search space expands. Recently, researchers have started to study the effects of initialization techniques other than conventional PRNGs on the performance of metaheuristics [Pant et al. 2009a]. These investigations revealed that many promising alternatives exist that can be used as population initializers. For example, some studies claim that alternative initialization techniques can increase the probability of finding global optima [Kimura and Matsumura 2005], reduce the variation of final outcome [Morrison 2003], decrease the computational costs [Kimura and Matsumura 2005] and improve the solution quality [Ma and Vandenbosch 2012]. Based on these findings, a vast and growing body of literature has been devoted to studying the other ways of generating initial populations.

3.2 Motivation

Correctly conducted literature reviews are foundations of any research projects. They naturally lead to new research ideas by drawing the big picture, categorizing the available solutions, and identifying the problems yet to be addressed. They often provide the research community actionable insights on the links between the knowns and unknowns in a field. On the other hand, the lack of such reviews is a hurdle for further studies in any domain. For example, researchers need to invest an immense amount of time in identifying open questions to address. They also need to study a vast number of publications to find state-of-the-art techniques to improve or baselines methods to compare with their solutions.

Although there exists a considerable number of publications regarding population initialization techniques in metaheuristics, little attention has been paid to summarize and analyze them comprehensively. This gap in the literature motivates us to conduct a systematical review of the existing population initialization techniques with a focus on efficient and scalable initializers to be applied to large-scale problems when the computational resources are scarce. In this chapter, we aim to highlight the importance and challenges of effective population initialization techniques for metaheuristics. It will help researchers working in this domain to understand the whole picture of the current studies in population initialization and facilitate them to choose suitable techniques in their research.

In particular, the contribution of this chapter is threefold:

1. Systematically compiling a comprehensive survey of population initialization techniques for use in metaheuristics by including almost all available methods. More precisely, the survey includes about 100 research papers published in a wide range of venues (refer to 3.3). The reviewed techniques exhibit a variety of features, such as different levels of stochasticness, compositionality, and generality.

2. Proposing a new approach for categorizing the population initialization techniques in a clear, concise, and systematic manner (see Figure 31). The result is a novel multifacet taxonomy that precisely classifies any available or future population initializer into a suitable category. Moreover, the taxonomy distinctly demonstrates the relationship between different types of initialization techniques.

3. Discussing the trends and open questions in this research topic and providing practical guidelines for future work. Highlighting the gaps in the literature while surveying the published researches is a crucial step to take before conducting further studies in the next chapters.

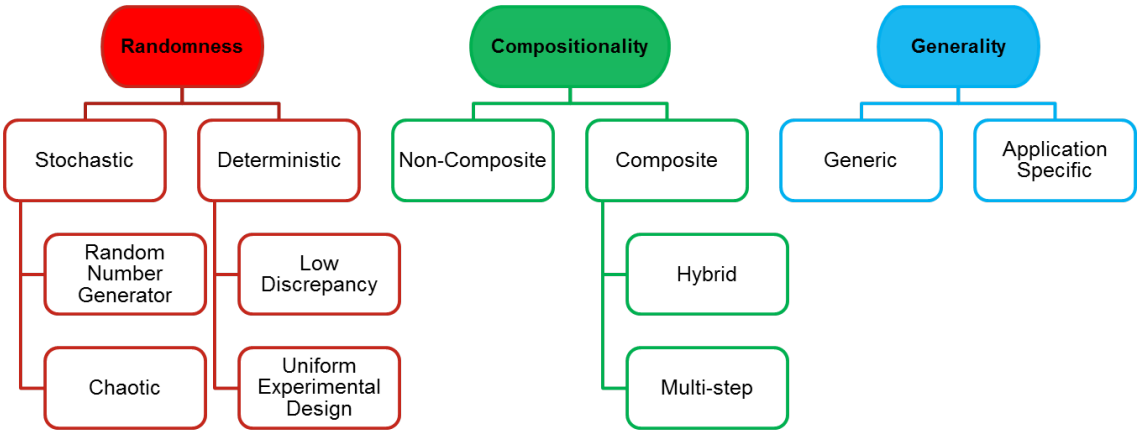


Figure 31: Three facets of categorizations for population initialization techniques: randomness, compositionality, and generality.

3.3 Population Initialization Techniques

Population initialization techniques are of multiple forms with a diverse range of attributes. Previously, very few attempts have been made to group these techniques into different categories. Although the existing taxonomies are informative, they suffer from two major problems. Firstly, most of them are not comprehensive enough to cover all types of current techniques. Secondly, they mostly categorize the methods from limited perspectives, *e.g.,* randomness, without considering other important factors that can exclusively characterize the techniques.

In this section, we propose a new approach in the categorization of population initialization techniques which, to the best of our knowledge, covers all of the existing population initialization techniques. The proposed categorization groups the existing techniques from three parallel perspectives that are easy-to-understand for general readers. These aspects are: *randomness*, *compositionality*, and *generality*. Within each of these three categories, we further demarcate representative techniques into several sub-categories and explain their shared attributes in details. The hierarchy of the proposed categorization is illustrated in Figure 31.

The three categorization criteria are determined based on two facts. Firstly, each criterion describes the technique from a unique and independent aspect. For example, whether a method is being random or not is entirely independent of its compositionality or generality status. This parallel classification helps us to describe a technique from three independent perspectives.

Secondly, each criterion is easy-to-understand for both metaheuristic researchers and practitioners. For example, if a researcher wants to choose an appropriate population initializer for a multi-start metaheuristic, the randomness of a population initializer must be taken into account. As another example, the generality of a technique plays an important role when a practitioner needs to find a proper initialization technique suitable for solving a very specific problem or a broad range of tasks (see Section 3.3.3 for further details).

The following subsections provide further details about each of the three main categories. The sub-categories with the advantages and disadvantages of the representative techniques are also discussed.

3.3.1 Randomness

From the randomness point of view, a sequence of numbers can be seen as *completely deterministic* to *truly random* [Dutang and Wuertz 2009]. While there is no universal agreement on the definition of randomness [Smith 2007], a truly random sequence is usually described as a sequence having strong properties such as complete *unpredictability*, *incompressibility* and *irregularity* [Ergün and Özoguz 2010]. Although several tools are available to measure these properties [Jun and Kocher 1999, Soto 1999], it is impossible to prove that a given sequence is truly random [Park and Miller 1988]. Instead, these tools can be employed as measures to determine that a given sequence is not truly random if it cannot pass all the tests [Park and Miller 1988]. Some of these tools may also be useful to determine whether a given sequence is *computationally random* or *statistically random* [Ma and Vandenbosch 2012]..

The results of the tests mentioned above (on unpredictability, for example) are susceptible to the power of the adopted tool [Dutang and Wuertz 2009]. To avoid such confusion we propose a simple, yet stable, alternative procedure here. We categorize the initialization techniques only according to their dependency on initial seeds. In other words, an initializer is considered as *stochastic* if it generates a different set of points when it is fed by different initial seeds. In contrast, techniques which continuously produce exactly the same set of points, regardless of different initial seeds, are counted as *deterministic*.

A. Stochastic Techniques

As discussed above, population initialization techniques which their outputs depend on initial seeds, are labeled as stochastic initializers. Here, we assume that the initial seed, which is provided by an external random source, is the only cause of randomness. The group of stochastic techniques can be divided into two subgroups: *Pseudo-Random Number Generators* and *Chaotic Number Generators*. Following paragraphs provide more details regarding these two subgroups.

Pseudo-Random Number Generator (PRNG): Due to the limitations of deterministic machines (*i.e.,* digital computers) in producing true random numbers [Jun and Kocher 1999], and also the lack of efficient techniques to sample random numbers from physical phenomena (*e.g.,* radioactive decay [Walker 2006] or atmospheric noise [Haahr 2010]), PRNGs are widely used in many applications to generate numbers which possess some irregularities and unpredictability attributes [Dutang and Wuertz 2009].

Generally speaking, PRNGs can be ranked based on two key factors: *cycle time* (*a.k.a.* period length) and *equidistribution*. In literature, the cycle time is defined as the smallest number of steps after which a PRNG starts regenerating the previously produced sequence. On the other hand, the equidistribution means all points in the range have equal frequency or probability of occurrence [Dutang and Wuertz 2009]. PRNGs which pass some tests (*e.g.,* DieHard [Jun and Kocher 1999, Soto 1999] and TestU01 [L'Ecuyer and Simard 2007]) can be considered as computationally or statistically random number generators.

By far, PRNGs are the most commonly used population initializers [Pant et al. 2009b, Rahnamayan et al. 2007a] in the metaheuristic literature. Among many variants, the Well-Equidistributed Long-period Linear (WELL) [Panneton et al. 2006], the KISS generator [Marsaglia and Zaman 1993], and Mersenne Twister [Matsumoto and Nishimura 1998] are the most widely used algorithms in this domain. The main properties which make these algorithms very common are their *simplicity* and *uniformity* which provides

efficiency, scalability, and effectiveness. Since fast PRNG tools are available in every programming language and there is no restriction on the number of points (*i.e.,* population size) and dimension size (*i.e.,* the number of decision variables), they can be easily applied to every problem. Moreover, where the dimensionality of the problem is not very high, and population size is large enough, PRNGs can provide initial populations with a satisfactory level of uniformity [Ma and Vandenbosch 2012]. As mentioned earlier, using an initial population with a high degree of uniformity can decrease the chance of missing a considerable part of search space through the optimization process.

The uniform populations generated by PRNGs can be easily transformed into nonuniform populations. Here, biased means the points in the population are not evenly distributed. In fact, they are scattered according to other statistical distributions such as Gaussian or Gamma distributions. Some previous works prefer to use nonuniform randomly generated points as initial population [Clerc 2008, Ma and Vandenbosch 2012, Pant et al. 2007]. This approach can be beneficial if the likelihood of finding the global optima is higher in some specific regions (*e.g.,* close to the center or some corners of the search space).

Apart from the useful properties of PRNGs, they suffer from the *curse of dimensionality* [Maaranen et al. 2004]. Indeed, PRNGs cannot produce perfect evenly distributed points [Helwig and Wanka 2008, Pant et al. 2007]. This drawback can be exacerbated when the search space dimensionality expands, or the population size is not sufficiently large.

To lessen the adverse effect of dimensionality on the performance of PRNGs, one may propose to increase the population size. However, as we empirically show in the next chapter, increasing population size while the computational budget is fixed cannot remedy the issue. In fact, blindly increasing the population size may result in early termination which in turn may cause the population not to converge at all. This can be even worse than a converged algorithm which missed a considerable portion of the search space due to poor uniformity of the initial population. Consequently, assuming uniformity as a critical factor of the initial population, metaheuristics need more effective techniques to generate initial points with better uniformity in high dimensions.

Chaotic Number Generator (CNG): Besides PRNGs, chaotic techniques are also widely employed to produce stochastic initial populations [Gao and Wang 2007, Gutiérrez et al. 2011, Dong et al. 2012, Gao and Liu 2012, Gao et al. 2012]. Technically, *Chaos theory* studies the behavior of dynamic systems which are very sensitive to their initial conditions. *Ergodicity* (*i.e.,* the ability to traverse all states in a particular region [Dong et al. 2012]), randomness and regularity are the primary attributes of chaotic systems [Zhang et al. 2009]. Since these features are desirable in many applications, several CNGs are proposed and broadly used [Senkerik et al. 2012, Pluhacek et al. 2013]. A proper mapping function is required to produce a chaotic population. In a very general form, a one-dimensional chaotic map can be defined as follows:

$$x_{i,j}^{k+1} := f_\rho(x_{i,j}^k) \tag{3.1}$$

where $x_{i,j}^k$ is the j^{th} variable of the i^{th} individual in the k^{th} iteration and ρ is the set of user-defined parameters. Generally speaking, the initial seed or $x_{i,j}^0$ is chosen randomly whereas the successive points (*i.e.,* $k > 0$) are produced by recursively applying the mapping.

To the best of our knowledge, no study has been done to comprehensively compare the uniformity of points that are generated by PRNG and CNG algorithms. However, it has been shown that adopting chaotic initialization techniques can improve the performance

of metaheuristics regarding population diversity, success rate, and convergence speed [Liu et al. 2005, Gutiérrez et al. 2011, Gao and Liu 2012].

Apart from all the advantages of CNGs, they suffer from a few limitations. Firstly, most of the previously proposed chaotic maps are designed for one, two, or at most three-dimensional spaces [Senkerik et al. 2013]. This means that the desirable attributes of the chaotic sequences may be visible in these low-dimensional projections, but it can hardly be generalized to higher dimensions. More studies are required to investigate the performance of high-dimensional populations which are generated using low-dimensional maps.

Secondly, the behavior of CNGs is very sensitive to the initial condition and their parameter settings [Liu et al. 2005, Senkerik et al. 2013]. For example, in Tent map with $\rho < 1$, the resulting population will converge towards 0 regardless of the initial seed. For $1 < \rho < 2$, however, all values close to 0 or $\rho/(\rho + 1)$ move away from them rapidly (but still remain in range $[0, 1]$). On the other hand, when $\rho > 2$ almost every point in the range $[0, 1]$ eventually diverge towards infinity. For Tent map, the only proper value for ρ which produces chaotic sequences is 2 [Dong et al. 2012].

Thirdly, the performance of CNGs is very sensitive to the precision level (or how the numbers are presented as binary sequences in the machines). As empirically studied in [Zelinka et al. 2013], different precision levels result in chaotic sequences with different periodicities which can significantly affect the optimizer's performance.

Fourthly, the existence of some attractors may cause the population to converge to a few fixed points. For example, in the case of the logistic map with $\rho = 4$ and an initial seed in the range $(0, 1)$, the four values of 0, 0.25, 0.5 and 0.75 are known as strong attractors [Yanguang et al. 2010]. The population must be checked against these attractors; otherwise, it may converge towards 0 after a few iterations (depending on the precision level).

Finally, it is not clear yet why in some cases a particular map performs considerably better than the others [Senkerik et al. 2013]. Unless the reason behind CNGs' performance is thoroughly and systematically investigated, general practitioners may face difficulties finding the best mapping function and parameter configuration.

B. Deterministic Techniques

As mentioned earlier, techniques which always generate the same set of points regardless of the initial seed are classified as deterministic initializers. In contrast with the stochastic methods, randomness and unpredictability are not essential objectives here [Uy et al. 2007]. Instead, deterministic initializers are specially designed to provide evenly distributed points in the entire search space. Recently, these techniques attract more attention considering, in the absence of prior knowledge about the problem, the uniformity of the initial population can enhance the exploration capacity of the optimizer in the early iterations [Chou and Chen 2000]. This may result in converging to a better solution regarding the objective value while saving a considerable amount of computational budget [Maaranen et al. 2004, Ma and Vandenbosch 2012].

In the literature, deterministic point generators are also referred as *low-discrepancy* techniques [Uy et al. 2007]. Literally, discrepancy means *nonuniformity*, and hence, discrepancy measures are the tools that we use to determine the level of nonuniformity in a given point set [Maaranen et al. 2004, Uy et al. 2007]. In other words, the point sets with small discrepancy scores are those with a high degree of uniformity. So far, two slightly different approaches for generating low-discrepancy sets are proposed: *quasi-random sequence* and *uniform experimental design* [Dutang and Wuertz 2009]. These techniques

are originally developed outside of the optimization domain; nevertheless, they have been adopted as initializer in metaheuristics.

Quasi-Random Sequence (QRS): The term *random* in the name of QRS should not confuse readers. These sequences are neither true random or pseudo-random. The QRS techniques, at least in the original form, are entirely deterministic methods and no random element (*e.g.,* random initial seed) is involved [Maaranen et al. 2007].

A bold advantage of QRS over the other techniques is that they have the support of theoretical upper-bounds on the discrepancy of the resulting sequences. Technically, when the population size is large enough (*i.e.,* $N \to \infty$), these limits indicate how much a particular QRS can be nonuniform in the worst case scenario [Uy et al. 2007]. The QRS techniques try to find the optimal parameter values to decrease the upper-bounds or to approach the lower-bounds [Dutang and Wuertz 2009]. Assuming positive correlation between the discrepancy of initial population and the objective value of the final solution (in minimization problems), one can select a proper QRS technique with the least discrepancy prior to running the optimizer. Since discrepancy calculation is computationally cheaper than executing an actual optimizer for several trials, having such theoretical upper-bounds is a valuable bonus for QRSs in comparison with the others.

Although QRS techniques have strong theoretical advantages over stochastic (and also other deterministic) methods, they suffer from some limitations. Firstly, the theoretical bounds on discrepancy may not be very beneficial in practice due to unsatisfied assumptions. In high-dimensional spaces, for example, population size is relatively smaller than what it should be to satisfy the underlying assumptions. Indeed, some studies raised doubt about QRS techniques having such superiority regarding the discrepancy over PRNGs in high dimensions [Morokoff and Caflisch 1994]. We will investigate this topic further in Chapter 4.

Secondly, various numerical algorithms for measuring discrepancy of a given sequence have been proposed [Wang and Sloan 2008]. These measures in some cases contradict each other. For example, a series may look more uniform than another sequence according to some discrepancy measures but less uniform regarding other discrepancy measures. This contradiction in discrepancy measures makes it difficult for general practitioners to compare QRSs to find the best technique before running the entire optimization process.

Finally, any correlation between discrepancies and solution's objective values has not been proven yet. Consequently, even finding a QRS initializer with the least discrepancy values might not result in the best final objective value after running the optimizer. These shortcomings can be some of the reasons behind the unpopularity of QRS in high-dimensional optimization.

Uniform Experimental Design (UED): The UED is a family of space-filling algorithms which tries to select a subset of available points that can be evenly scattered across a given range. Since its inception in the 1980s, it has been widely used in industrial and computer simulation designs [Fang and Lin 2003]. For some low-dimensional spaces, UED tables have been calculated and published such that practitioners do not need to execute the generative algorithms.

Suppose we created a complete grid in a D-dimensional space which each variable has exactly q different values or levels. The total number of points in the grid (*i.e.,* population size) would be q^D. In theory, having large enough q results in a perfectly uniform population. However, evaluating such big population is practically impossible even for small-scale problems. To lessen complexity, UEDs can be used to systematically select a smaller num-

ber of points from the full grid which still preserve the uniformity of the original grid to a great extent. So far, a wide range of UEDs such as *uniform design* [Peng et al. 2012] and *orthogonal design* (OD) [Leung and Wang 2001] has been employed as population initializer in iterative optimizers.

In comparison with QRSs, the UEDs have two main advantages. Firstly, QRSs only consider one-dimensional projection uniformity; while non-orthogonal and orthogonal UEDs consider two and D-dimensional (in addition to one-dimensional) projection uniformity [Fang and Lin 2003]. These extra considerations can provide more desirable regularity and uniformity. Secondly, UEDs usually generate discrete points while QRSs are initially designed for real-value spaces. This unique attribute helps UEDs to be directly applicable to nominal and discrete optimization problems.

The UEDs also have some limitations. Firstly, the performance of many UEDs depends on the parameter settings. In orthogonal design [Gong et al. 2010], for example, the number of levels (*i.e.,* q) plays a significant role. While large values of q can result in a more uniform population, they can exponentially increase the population size. This enormous population size prevents users from using the orthogonal design technique directly on moderate to large-scale problems. To remedy this problem, [Peng et al. 2012] suggests to evaluate all generated points and then pick the best subset according to their objective values. This solution potentially wastes a considerable portion of the computational budget in the early stage while it could be used in the course of optimization.

In contrast, [Leung and Wang 2001] suggests group variables using some heuristics and use the same values for all variables in each group. This solution can reduce the resulting population size, however dramatically degrades the uniformity of population (which contradicts the original goal of preserving the uniformity of the population). Furthermore, the variable grouping forces extra computational costs to the optimization algorithm.

3.3.2 Compositionality

In the context of population initialization, we explain the compositionality as a measure of the number of standalone procedures that are involved in a technique. Based on this criterion, population initialization techniques fall into *composite* and *non-composite* groups. In the following paragraphs, we provide more details and some examples for each group of techniques.

A. Noncompositional Techniques

From the compositionality point of view, all basic techniques which produce a population of potential solutions in only one single step are labeled as a noncompositional algorithm. Hence, regardless of being stochastic, deterministic, generic or application specific, as long as a technique cannot be divided into disjoint population initialization techniques, it is considered as a noncompositional algorithm. Therefore, all methods which were reviewed in Subsection 3.3.1 are noncompositional unless one combines two or more of them.

B. Compositional Techniques

In contrast with the noncompositional group, techniques which comprise of more than one stage are labeled as compositional algorithms. The methods in this family can be further demarcated into two subgroups: *hybrid* and *multi-step* techniques.

Hybrid Compositional Initializers: Each component of a hybrid technique can be separately employed as an independent noncompositional initializer. For example, while

each of CNG and PRNG techniques can be used as individual population initializers, one may utilize a CNG to generate the initial seed for a PRNG, or vice versa.

Another example of hybrid initialization techniques which have been used in several studies is those that try to bring some randomness to QRS techniques. This way, the resulting population may benefit from both the uniformity feature of QRS populations and the randomness attribute of PRNG methods. Based on the employed hybridization techniques, the resulting algorithms may be different in titles and characteristics. The *random start* QRS [Uy et al. 2007], *scrambled* QRS [Kimura and Matsumura 2005, Owen 1995, Dick and Pillichshammer 2005] and *mixed pseudo-quasi-random sequence* [Dutang and Wuertz 2009] are some examples of this category of initializers.

In theory, hybrid techniques inherit the advantages and disadvantages of the elementary methods which they are constructed from. Consequently, studying the constructive components can shed more light on hybrid techniques as well. On the other hand, when our knowledge about the basic ingredients (*i.e.,* noncompositional techniques) is insufficient, studying hybrid methods would provide little benefit and interest.

Multi-step Compositional Initializers: As opposed to hybrid techniques, a multi-step technique comprises two or more components which at least one of them cannot be employed as a standalone initializer. In other words, multi-step procedures generally process and refine the previously generated population in later steps. One of the most popular multi-step techniques which is widely used in different algorithms and applications is the family of *opposition based learning* (OBL) techniques [Rahnamayan et al. 2006; 2007a; 2008, Gao et al. 2012].

In the first step, OBL techniques generate a set of points called *original population*. The original population can be generated using any initializer technique (*e.g.,* PRNG [Rahnamayan et al. 2006], CNG [Gao et al. 2012] or UED [Peng and Wang 2010]). Then, some simple heuristic rules are employed to produce another population of the same size in the second step. This new population is generally referred as the *opposite population*. At the final stage, a subset of the union of both populations is selected based on their fitness values. Equation 3.2 shows the heuristic rule which produces an opposite population based on the original population:

$$\tilde{x}_{i,j} = a_j + b_j - x_{i,j}, \quad j = 1, ..., D. \tag{3.2}$$

where $X_i =< x_{i,1}, x_{i,2}, ..., x_{i,D} >$ is the i^{th} individual of the original population and each variable $x_{i,j}$ is bounded by (a_j, b_j) such that $a_j \leq x_{i,j} \leq b_j \quad \forall i \in \{1, \ldots, N\}$.

Several variations of OBL techniques have been proposed, so far [Wang et al. 2011, Al-Qunaieer et al. 2010, Ergezer and Sikder 2011]. In *quasi-opposition based learning* (QBL), for example, quasi-opposite points are used instead of the actual opposite points [Rahnamayan et al. 2007c]. A quasi-opposite point is a randomly generated point located between the opposite point and the middle point (*i.e.,* $a_j + (b_j - a_j)/2$ for $j = 1, ..., D$). More information on other variants of OBL techniques such as quasi-reflection opposition-based learning [Ergezer et al. 2009], center-based sampling [Rahnamayan and Wang 2009], generalized opposition-based learning [Wang et al. 2009] and current optimum opposition-based learning [Xu et al. 2013] is available in [Ergezer and Sikder 2011] and [Xu et al. 2014].

According to the probability theory, there is 50% chance for an unknown solution to be closer to the opposition point than the original point [Rahnamayan et al. 2007a]. In [Rahnamayan et al. 2007c], the authors proved that the points generated using QBL

have even higher chance to be closer to an unknown solution than points produced by the previous OBL technique.[1]

Besides the OBL initializers, a few other multi-step techniques also use fitness function as a guideline for enhancing the initial population. Indeed, exploiting objective function to gain some knowledge about the fitness landscape is very common in the initialization step [Dasgupta et al. 2009a]. For example, [Zhang et al. 2011] propose an initializer that exploits local and global selection mechanisms to generate high-quality initial population for job-shop scheduling. In [Kumar et al. 2013], authors suggest applying a hill-climbing local search to improve initial population quality. More advanced searches such as quadratic interpolation [Pant et al. 2009a], nonlinear local search (*a.k.a.* simplex) [Parsopoulos and Vrahatis 2002, Ali et al. 2012], centroid-based sampling [Khanum and Jan 2011], Tabu search [Sharma and Tyagi 2013] and smart sampling [de Melo and Botazzo Delbem 2012b] are also used as the second steps of some compositional initialization techniques.

Although the multi-step techniques achieved excellent results, they suffer from three main shortcomings; Firstly, these algorithms consume a part of the computational budget to evaluate the fitness function and select the best subset of both populations. One remedy could be employing surrogate models to reduce the computational costs. However, given the small size of the training set at the early steps and the dimensionality of the search space, the accuracy of the subset selection might not be very high. Secondly, since these techniques calculate the secondary points based on the original population, their performances to a great extent depends on the quality of the original points. Accordingly, some studies proposed to use more advanced point generators for producing the original population rather than simple PRNGs [Gao et al. 2012, Peng and Wang 2010]. Finally, because of the greediness of the selection mechanism in most of these techniques, the chance of losing informative building blocks at the first stage is very high. In other words, it is very likely that some individuals which have useful components are immediately excluded only due to their relative low fitness values.

The Centroidal Voronoi Tessellation (CVT) is another example of multi-step methods [Richards and Ventura 2004, Saka et al. 2007] in a sense it does not use fitness function, but other metrics to enhance initial population quality. The CVT tries to partition search space into subspaces with equal volumes using algorithms such as Lloyd's technique [Lloyd 1982]. The partitions' centroids then will be used as initial population of the optimizer. In the simplest form, a temporary population is generated using a PRNG or any other technique. Then, with the aid of many randomly generated auxiliary points, the search space is divided into N partitions. These partitions and their centers are iteratively enhanced until the termination criterion is met. At the final stage of the algorithm, the CVT returns the partitions' centroids as the initial population of the metaheuristic [Richards and Ventura 2004]. Similar to CVT, *simple sequential inhibition process* (SSI) is also used in a few studies to produce evenly scattered populations [Maaranen et al. 2007].

In comparison with the other multi-step techniques that we reviewed, CVT and SSI have two main advantages. Firstly, these algorithms can produce geometrically evenly scattered populations without using any objective function evaluations, whereas the others need several function evaluations. Secondly, since CVT and SSI do not select points based on their fitness values, it is less likely to miss a significant part of search space as the greedy selection mechanism in some other multi-step techniques sometimes does.

However, CVT also suffers from some limitations. Firstly, both CVT and SSI are known as computationally expensive techniques. To lessen this shortcoming, one can

[1] The OBL techniques is not only used in the initialization step but also employed as general sampling techniques in the course of optimization.

adopt a QRS or UED technique to generate the temporary population in the first step and increase the convergence speed of the whole pipeline. Secondly, their performance depends on the internal partitioning (or clustering) algorithm and the employed distance measures. Accordingly, practitioners must choose extra parameters (*e.g.,* distance measure and stopping criterion) in addition to the optimizers' settings. Finally, these iterative techniques might not converge when the population size is relatively small. This situation is more likely to happen when dealing with high-dimensional optimization problems.

3.3.3 Generality

In the context of population initialization, the generality of an algorithm refers to the variety of the domains that it can be applied to. In terms of generality, population initialization techniques are grouped into two categories: *generic* and *application-specific* techniques.

A. Generic Techniques

The population initialization techniques which can be directly applied to all types of optimization problems are called generic methods. In this sense, all techniques described in the previous sections belong to the generic category. These techniques assume that the given optimization problem is a black-box puzzle. Therefore, no precise knowledge about the region of interest or building blocks of the potential solution is available before running the optimizer. In the absence of such prior knowledge, generic population initialization techniques can be used efficiently and effectively.

B. Application-Specific Techniques

The application-specific group comprises a few techniques which are specially designed to be applied to particular real-world problems. In the design of such procedures, inventors exploit domain knowledge to avoid searching unnecessary regions, producing more promising results and boosting metaheuristics convergence speed. Application-specific techniques are potentially beneficial in solving problems that they are specially designed for. However, they may not be useful, efficient or even applicable in many other areas. Consequently, studying these techniques must be only done by the experts in those specific domains. Table 31 presents some of the previously published studies on the application-specific population initialization techniques.

3.4 Chapter Summary

In this chapter, we reviewed the majority of the available population initializers techniques that have been used in the context of population-based metaheuristics regardless of the size of the problem in hand. We provided a new three-facet categorization schema that covers all available and potential future initialization techniques. Studying this taxonomy helps us to identify the potentials and pitfalls of each family of initializers, as well as the similarity and dissimilarity of the surveyed techniques. In the next chapter, we investigate whether these population initializers are effectively scalable to higher dimensions by thoroughly studying a few representatives from each category.

Table 31: Application-Specific Population Initialization Techniques

Application	**Reference**	**Year**
Breast cancer prognosis	[García-Arnau et al. 2007]	2007
Flexible job-shop scheduling	[Pezzella et al. 2008]	2008
p-median problem	[Li et al. 2011]	2011
Two-stage stochastic mixed-integer	[Tometzki and Engell 2011]	2011
FSS and antenna arrays	[Gutiérrez et al. 2011]	2011
Flexible job-shop scheduling	[Zhang et al. 2011]	2011
Antenna design	[Ma and Vandenbosch 2012]	2012
Circle detection	[Dong et al. 2012]	2012
Segmentation	[Guerrero et al. 2012]	2012
Grid Scheduling	[de Albuquerque Torreão and Vimieiro 2018]	2014
Video game playing	[Gaina et al. 2017]	2017
Smart community optimization	[Sato and Fukuyama 2017]	2017
Stochastic volatility modeling	[Osvald et al. 2017]	2017
Steel trusses optimization	[Kazemzadeh Azad 2018]	2018
Subgroup discovery	[de Albuquerque Torreão and Vimieiro 2018]	2018
Electric Motor Design	[Essaid et al. 2018]	2018

CHAPTER 4

Scalability of Population Initializers

In Chapter 3, we reviewed and categorized the major families of population initialization algorithms that have been employed in metaheuristics. According to literature, some of these techniques can boost the convergence speed of the optimization algorithms which can save a considerable portion of the precious computational budget. Undoubtedly, the effective use of resources is more critical in large-scale optimization applications. In this chapter, we aim to study the scalability of these underrepresented initialization techniques. More specifically, we investigate whether the application of alternative initializers in high-dimensional spaces has any significant benefit over the use of the conventional pseudo-random number generators. To achieve this goal, we select a few representatives from each of the categories discussed in the previous chapter. Then, we study the effect of dimensionality on their performance while a commonly used initializer is used as the control method.

4.1 Motivation

As discussed in Chapter 3, pseudo-random number generators (PRNGs)[1] are widely used as the population initializer in many iterative metaheuristic algorithms. Although a significant body of literature has proposed several new ways of generating initial populations, only a very little attention has been paid to the potential influence of these techniques on the performance of optimizers. Several studies have claimed that alternative methods can substantially increase the probability of finding the optimum solution, dramatically decrease the computational cost [Kimura and Matsumura 2005], reduce the variation of results in different runs [Morrison 2003], and significantly improve the solution quality [Ma and Vandenbosch 2012].

While the application of alternative initialization techniques in optimization repeatedly reported as promising, they have been mainly used to solve low or medium size problems. They have not been used widely to deal with large-scale problems which usually have thousands of decision variables. For example, all participants of competitions on large-scale global optimization (LSGO) held at the IEEE Congress on Evolutionary Computation (CEC-2008, CEC-2010, and CEC-2012), except [Tseng and Chen 2008], only used conventional PRNGs in the initialization step. Considering the ever-growing demands on solving large-scale problems, it is essential to investigate whether advanced initializers can improve the performance of metaheuristics.

[1] We refer to non-PRNG algorithms as alternative initializers.

Another important motivation for conducting this study is that the computational budget is always limited and managing these precious resources is even more vital in the large-scale optimization. If the claimed advantages of adopting the alternative techniques (*e.g.,* enhancing the convergence rate) can be achieved in the high-dimensional spaces, then one can save an exceptional portion of the budget by merely substituting the conventional PRNG point generator with a more effective algorithm.

As per the literature, a few comparative studies have been previously carried out on the effect of different initialization methods on the optimization performance [Clerc 2008]. Although these studies are scientifically informative, they suffer from several shortcomings. Firstly, to the best of our knowledge, they are limited to investigate just a few methods mostly from the same category of initializers. For example, our literature review revealed that they compared very few (*i.e.,* four or less) initialization methods. Secondly, all of the available comparison studies only examined low dimensional problems. For instance, none of the previously published researches studied any problem with more than 60 decision variables [Kimura and Matsumura 2005]. Thirdly, the past studies seldom investigated the relation between the initialization method and other related parameters such as objective function dimensionality and population size. Finally, a number of them only studied the effect of initialization methods on a small set of particular problems. For example in [Ma and Vandenbosch 2012], the impact of different random number generators are just reviewed on the design of a specific type of antenna micro-strip arrays.

These previously published studies and the derived conclusions, although valuable, cannot be generalized well to problems of very different nature. In this chapter, however, we address these limitations and provide a more comprehensive comparative study on the influence of population initialization size and techniques on a variety of large-scale problems.

4.2 Background

To study the scalability of initialization techniques, we select eight population initialization techniques across multiple categories. These algorithms exhibit different attributes such as being compositional or noncompositional and producing deterministic or stochastic sets of points. However, all of the chosen algorithms are general-purpose (*i.e.,* generic) initializers that can be applied on a wide range of problems (see Chapter 3 for further information about the categories and attributes). In the following, we briefly review the chosen initializers.

Mersenne Twister: By far, the Mersenne Twister is the most widely used equidistributed uniform pseudo-random number generator [Matsumoto and Nishimura 1998]. We adopt this technique from the stochastic and noncompositional categories (see Chapter 3 for more details). We will refer to this algorithm as RNG and will use it as the baseline method. The RNG technique and is dimension agnostic which means it can only produce a sequence of one-dimensional numbers which we form them as vectors or matrices of other dimensions when we need. Therefore, it only accepts one parameter which is the initial seed. For our experiments, we use the *rand* function implemented in Octave version 3.6.2.

Chaotic Tent Map: This technique, which we call it TNT hereafter, is a chaotic sequence generator that uses Tent mapping function to calculate the next number based on the previous point. The algorithm's steps are straightforward:

1. Select or randomly generate the first number as $0 \leq x_{i,j}^{(0)} \leq 1$ where $x_{i,j}$ is the j^{th} variable of the i^{th} potential solution.

2. Do the following K times: $x_{i,j}^{(k+1)} = \rho(1 - 2|x_{i,j}^{(k)} - 0.5|)$ where k and ρ indicate the iteration number and bifurcation factor, respectively [Dong et al. 2012].

For a better scattered initial population, one can substitute k by j (or as a function of j). Using a random initial seed at Step 1 and reasonable values for K and ρ in Step 2 will result in a stochastic population with chaotic attributes.

Good Lattice Points: The good lattice point, or GLP for short, is a noncompositional deterministic point generator which produces low-discrepancy sequences [Zaremba 1966]. In this study, we follow the implementation of GLP that is proposed in [Sloan and Joe 1994] and has been widely used in Monte Carlo integrations [Ebert and Kritzer 2018]

Sobol's Set: This is another well-known quasi-random sequence generator that is broadly used to produce low-discrepancy sets of numbers [Maaranen et al. 2004]. By definition, Sobol's algorithm, which we will refer to as SBL hereafter, is a noncompositional deterministic technique. In these series of experiments, we use the official Matlab's (version 7.11) implementation of SBL as proposed originally in [Bratley and Fox 1988].

Uniform Experiment Design: This is a noncompositional deterministic space-filling technique that follows regular patterns to guarantee evenly distributed set of points. Since the points generated by the uniform experiment design algorithm (UD) are made of discrete numbers, some post-processing is required to map the values to the desired ranges (*i.e.,* within the decision variables' bounds). In this chapter, we follow the implementations of UD given by [Peng et al. 2010] and [Peng et al. 2012].

The UD works as follows: suppose we want to generate a uniformly scattered set of size N in D dimensional hypercube. Also, let $P = \{p_1, p_2, ...p_D\}$ be D randomly selected prime numbers smaller than N, then $X_{i,j} = (i \times p_j)\ mod\ N$ is the jth variable of ith initial individual. Note that the number of prime numbers fewer than N should be larger than or equal to D. Otherwise, we have to increase the population size.

Orthogonal Experiment Design: The orthogonal design (OD) is another noncompositional deterministic space-filing method that has been applied to produce evenly scattered points over the search space [Leung and Wang 2001]. Technically, OD produces an orthogonal 2D array like $L_N(Q^D)$ where N is the number of rows (*i.e.,* population size), D is the number of columns (*i.e.,* factors or dimension size), Q is the levels, and L denotes Latin square. The value of Q should be an even integer and $N = Q^J$ while J is a positive integer satisfying $D = \frac{Q^J - 1}{Q - 1}$. The main attributes of an orthogonal array are as follows:

1. For the factor in any column, every level occurs exactly $\frac{N}{Q}$ times. This attribute makes the resulting population very uniform.

2. Orthogonality of the array is not sensitive to the number or the order of columns. Therefore, one can reorder the columns or remove a number of them, and the resulting array is still orthogonal.

This study follows the implementation of OD presented in [Leung and Wang 2001].

Opposition-Based Learning: The broad family of opposition-based learning population generator consists of several compositional (multi-step) techniques that generate a new set of samples called *opposite population* based on a mapping function and a given set of points (*a.k.a. original population*). In this chapter, we study two variants of these methods namely opposition-based learning (OBL) and quasi-opposition-based learning (QBL).

The OBL works as follows: Let $\mathbf{X}_i = \langle x_{i,1}, x_{i,2}, ..., x_{i,D} \rangle$ be the i^{th} individual in the population where each variable $x_{i,j}$ is bounded by $\{a_i, b_j\}$. The opposition point is defined as $\tilde{\mathbf{X}}_i = \langle \tilde{x}_{i,1}, \tilde{x}_{i,2}, ..., \tilde{x}_{i,D} \rangle$ where:

$$\tilde{x}_{i,j} = a_j + b_j - x_{i,j}, \quad j = 1, ..., D. \tag{4.1}$$

Note that we generate the original population $\mathbf{X}$ randomly and then the opposition population $\tilde{\mathbf{X}}$ based on Equation (4.1). Now, we merge both populations and evaluate them based on the objective function. Finally, we select the top N individuals according to their fitness values to form the initial population for the downstream optimizer.

The QBL algorithm works similar to the OBL technique. The only difference between these two is that QBL uses a slightly different mapping to produce the opposition (or more precisely the quasi-opposition) population. Considering the definition of opposite points in Equation (4.1), the quasi-opposition vector of $\mathbf{X}_i$ is $\breve{\mathbf{X}}_i = \langle \breve{x}_{i,1}, \breve{x}_{i,2}, ..., \breve{x}_{i,D} \rangle$ where:

$$\breve{x}_{i,j} = \begin{cases} rand(m_j, \tilde{x}_{i,j}) & \text{if } x_{i,j} \leq m_j \\ rand(\tilde{x}_{i,j}, m_j) & \text{if } x_{i,j} > m_j \end{cases} \tag{4.2}$$

where $m_j = \frac{b_j - a_j}{2}$ and $rand(\alpha, \beta)$ is a random number drawn uniformly from (α, β) range. Similar to OBL, after the calculation of $\breve{\mathbf{X}}$, we merge both original and quasi-opposition populations into one big population of size $2 \times N$. Then, we select the N fittest solutions based on the given fitness function.

4.3 Experiments Setup

This section comprises two sets of experiments. In the first part, we choose three different dimension sizes (*i.e.,* 100, 500, and 1000) to assess the scalability of the eight chosen initializers on the final results of the adopted optimizer. More precisely, we investigate whether an initialization technique that outperforms RNG as the baseline on mid-scale problems (*i.e.,* 100 dimensions) can also improve it in higher dimensions (*i.e.,* 500 and 1000-dimensional problems). In this part, all parameters settings including the population size are kept constant for all methods and cases.

In the second part, we examine the performance of the employed optimizer only on problems with 500 decision variables while varying the population size. This set of experiments aims to investigate whether increasing the population size can improve the relative performance of the control method. In other words, we expect a promising initializer to outperform the baseline regardless of the chosen population size. Therefore, we compare the alternative techniques with six different population sizes. Note that, apart from the population size, other parameters are kept the same as the first part of the experiments.

We use CEC'08 Large-Scale Global Optimization (LSGO) benchmark functions in both sets of experiments [Tang et al. 2007]. An overview of the studied benchmark set is provided in Table 41. For all experiments, we use standard implementation of Differential Evolution (DE/local to best/1/bin) [Storn and Price 1997]. Among all the available metaheuristics models, we select DE because of its simplicity and popularity in solving

Table 41: The CEC 2008 LSGO Benchmark Functions

	Basis Function	**Modality**	**Separability**
f_1	Shifted Sphere Function	Unimodal	Separable
f_2	Shifted Schwefel's Problem 2.21	Unimodal	Nonseparable
f_3	Shifted Rosenbrock's Function	Multimodal	Nonseparable
f_4	Shifted Rastrigin's Function	Multimodal	Separable
f_5	Griewank's Function	Multimodal	Nonseparable
f_6	Shifted Ackley's Function	Multimodal	Separable

Table 42: The parameters and their values for *DE/local to best/1/bin*

Parameter	**Value**	**Parameter**	**Value**
Strategy	local to best/1/bin	Bound Constraints	None
Crossover Rate	0.90	Termination Criteria	Func. Eval.
F Weight	0.85	Max. Func. Eval.	$5000 \times D$

large-scale problems [Omidvar et al. 2010a]. We adopt the typical values for DE parameters to keep the experiments reproducible and straightforward (we further study these variables in Chapter 5). The parameters and their values are presented in Table 42.

Note that since in this chapter we mainly focus on the initialization step and its effect in saving the computation budget, the advanced DE variations such as those using self-adaptive parameter tuning [Qin et al. 2009, Yang et al. 2008c] or cooperative coevolution versions [Omidvar et al. 2010b;a] are avoided to reduce the effect of other parameters.

4.4 Results and Discussion

We run each of the aforementioned eight population initializers on six minimization problems from CEC'08 LSGO benchmark set. Table 43 presents the result of the first set of experiments The table contains the average values of 50 independent trials of running DE on 100, 500, and 1000-dimensional functions. Since this series of experiments aim to compare the potential improvement in final solution quality by employing more advanced initialization techniques, RNG is selected as the baseline method. Consequently, all initialization methods are only compared with RNG.

In all cases, the methods which significantly outperformed RNG (as the control technique) are highlighted using a **boldface** font. The algorithms which their outcomes are statistically similar to RNG are written in *italic* style. The initializer with the best results for each function is also emphasized with an asterisk symbol "*" only if that technique significantly enhanced the RNG results. For all the presented cases, the term *significant* means Wilcoxon rank-sum test rejects the null hypothesis with 95% confidence level. On the other hand, the phrase *statistically similar* in this context means Wilcoxon rank-sum test does not reject the null hypothesis. The null hypothesis in this test is that RNG results and its competitors' results represent the same statistical distribution.

Table 43: Mean of 50 independent runs ($N = 50$)
The stared*, **bold**, or *italic* numbers denote best methods, significantly better results or statistically similar results, respectively.

D	f	**RNG**	**TNT**	**SBL**	**GLP**	**UD**	**OD**	**OBL**	**QBL**
100	f_1	7.49e−21	**4.26e−21**	**6.80e−21**	**6.15e−21**	**7.41e−21**	*5.30e−21*	**5.97e−21**	**4.04e−21***
	f_2	7.74e+00	**4.54e+00***	**6.68e+00**	*6.35e+00*	7.78e+00	*5.58e+00*	**7.58e+00**	**6.32e+00**
	f_3	1.03e+02	*9.64e+01*	*9.95e+01*	*9.95e+01*	1.04e+02	*9.78e+01*	1.05e+02	**9.73e+01**
	f_4	2.55e+02	*2.56e+02*	*2.60e+02*	*2.59e+02*	*2.56e+02*	*2.59e+02*	*2.61e+02*	*2.58e+02*
	f_5	0.00e+00	*0.00e+00*	*0.00e+00*	*0.00e+00*	*0.00e+00*	*0.00e+00*	*0.00e+00*	*0.00e+00*
	f_6	1.70e+01	**1.71e−09**	**2.78e−11**	**4.72e−11**	1.99e+01	*1.11e−11*	**1.14e−10**	**9.18e−12***
500	$f1$	2.60e−08	**1.10e−08***	**2.34e−08**	*2.36e−08*	2.88e−08	3.70e−08	**2.55e−08**	**1.57e−08**
	$f2$	8.28e+01	**7.57e+01**	*8.09e+01*	*8.00e+01*	**7.99e+01**	*7.37e+01*	**7.97e+01**	**7.83e+01**
	$f3$	6.64e+02	*6.53e+02*	*6.53e+02*	*6.73e+02*	*6.82e+02*	*6.55e+02*	*6.95e+02*	*6.69e+02*
	$f4$	2.54e+03	*2.63e+03*	*2.55e+03*	*2.56e+03*	*2.57e+03*	2.56e+03	2.55e+03	*2.56e+03*
	$f5$	3.22e−09	**1.25e−09***	**2.86e−09**	*2.75e−09*	3.57e−09	4.65e−09	**3.03e−09**	**1.84e−09**
	$f6$	9.65e+00	**9.15e−06**	**1.02e+00**	**2.76e+00**	1.33e+01	2.10e+01	**5.80e+00**	**7.43e−06***
1000	$f1$	1.79e-09	**1.09e-09***	**1.57e-09**	*1.59e-09*	2.77e-09	2.37e-09	**1.75e-09**	**1.24e-09**
	$f2$	1.18e+02	**9.50e+01**	**9.48e+01***	*9.49e+01*	1.22e+02	**1.01e+02**	**1.16e+02**	**1.15e+02**
	$f3$	2.29e+03	*2.33e+03*	*2.26e+03*	*2.37e+03*	*2.31e+03*	*2.22e+03*	*2.29e+03*	*2.32e+03*
	$f4$	7.18e+02	1.23e+03	7.59e+02	*7.46e+02*	1.06e+03	**7.17e+02***	7.29e+02	*7.92e+02*
	$f5$	1.18e-10	*8.34e-11*	*1.12e-10*	*1.10e-10*	1.81e-10	1.60e-10	1.18e-10	**8.11e-11***
	$f6$	7.40e-01	**2.80e-06**	**3.75e-05**	**1.30e-05**	3.81e+00	2.12e+01	**1.54e-05**	**1.97e-06***

From the first six rows of Table 43 (*i.e.,* $D = 100$), it is apparent that except the OD, all other methods outperformed the RNG's results in several cases. The figure that we present in this table indicates that all methods can solve f_5 successfully and the performances of all techniques on f_6 are statistically similar. Based on these outcomes, we conclude that in these cases, employing alternative methods has no meaningful advantages over the conventional RNG. In the other cases (*i.e.,* f_1, f_2, f_3 and f_4), however, using alternative methods is notably advantageous. This part of the table also shows that most of the methods (*e.g.,* TNT, SBL, GLP, OD, and QBL) can be considered as alternatives of RNG because their results are either significantly better or at least statistically similar to the RNG's performance.

The figures in the other two parts of Table 43 (*i.e.,* $D \in \{500, 1000\}$) are more or less similar to the top part. Although there is no significant improvements in the case of f_3 in 500 and 1000 dimensions and f_4 in 500 dimensions, the alternative techniques considerably boost DE final results in the remaining cases (*i.e.,* f_1, f_2, f_5, and f_6). These three tables confirm that even in very complex problems (*i.e.,* nonseparable, multi-modal, and large-scale), initialization methods can significantly improve the quality of the outcome. In other words, these results suggest that practitioners may choose some of these alternatives over the usual case of RNG for population initialization purposes when dealing with large-scale and complex search landscapes.

Table 44: The summary of the scalability study ($D \in \{100, 500, 1000\}$, $N = 50$)
The values in the W-T-L triplets denote how many times an alternative initializer is significantly better than RNG, statistically similar to RNG, or significantly worse than RNG, respectively.

D	**TNT**	**SBL**	**GLP**	**UD**	**OD**	**OBL**	**QBL**
100	3−3−0	3−3−0	2−4−0	1−2−3	0−6−0	3−2−1	4−2−0
500	4−2−0	3−3−0	1−5−0	1−2−3	0−2−4	4−1−1	4−2−0
1000	3−2−1	3−2−1	1−5−0	0−1−5	2−1−3	3−1−2	4−2−0
Sum	10−7−1	9−8−1	4−14−0	2−5−11	2−9−7	10−4−4	12−6−0
%	56−38−6	50−44−6	22−78−0	11−28−61	11−50−39	56−22−22	67−33−0

Table 45: The summary of the case studies on varying population sizes. ($N \in \{50, 100, 150, 200, 250, 300\}$, $D = 500$)
The values in the W-T-L triplets denote how many times an alternative initializer is significantly better than RNG, statistically similar to RNG, or significantly worse than RNG, respectively.

N	**TNT**	**SBL**	**GLP**	**UD**	**OD**	**OBL**	**QBL**
50	4−2−0	3−3−0	1−5−0	1−2−3	0−2−4	4−1−1	4−2−0
100	6−0−0	5−1−0	4−2−0	0−0−6	0−4−2	6−0−0	6−0−0
150	6−0−0	6−0−0	4−2−0	3−0−3	0−5−1	5−0−1	5−1−0
200	6−0−0	6−0−0	4−2−0	2−0−4	0−6−0	5−0−1	5−1−0
250	6−0−0	5−1−0	5−1−0	2−0−4	0−6−0	6−0−0	5−1−0
300	6−0−0	5−1−0	5−1−0	3−0−3	0−5−1	6−0−0	6−0−0
Sum	34−2−0	30−6−0	23−13−0	11−2−23	0−28−8	32−1−2	31−5−0
%	94−6−0	83−17−0	64−36−0	30−6−64	0−78−22	89−3−8	86−14−0

We also include Table 44 for a more clear comparison between the performance of alternative initialization methods on problems with different dimension sizes. For each of the techniques and dimension sizes, a triplet of $W-T-L$ is given (the W, T, and L stand for wins, ties, and loses, respectively). For a particular technique, the value of W denotes the number of cases that it could significantly improve the baseline. The T value represents the number of functions which the results of this technique were statistically similar to the control method's results. And finally, the L indicates how many times this alternative initializer performed significantly worse than the RNG (all according to aforementioned Wilcoxon rank-sum statistical test). For example, consider the case of OBL in 100 dimensions. In this case, the WTL triplet is $3-2-1$ which means that the OBL significantly excels RNG on three functions, performs statistically similar to RNG on two problems, and in only one case, it produces results which are considerably worse than the RNG outcomes.

The Table 44 is very informative in several ways. Firstly, comparing each method's performance in different dimension sizes indicates that the relative ranks of the alternative techniques (except OD) are very consistent. For instance, TNT significantly outperforms RNG in at least 50% of functions in every dimension sizes while GLP is never significantly surpassed by RNG in any dimension size. Another striking observation is that, regardless of the dimensionality, the TNT, SBL, OBL, and QBL can significantly outperform RNG in at least 50% of the studied problems. QBL is also found to be the best method among these eight initialization techniques whereas both GLP and QBL can be considered as strong candidates to substitute RNG on large-scale problems because they are never significantly worse than the RNG.

Having discussed the effects of the selected initialization methods on large-scale problems with different dimension sizes, now we want to analyze the performance of these algorithms when population sizes are varied. Figure 42 illustrates the median values of 50 independent trials of all eight initializers with six different population sizes: $N \in \{50, 100, 150, 200, 250, 300\}$.

Figure 42 demonstrates that except for GLP, the performance of DE decreases when population size is increased. This result is expected because the computational budget is kept fixed for all population sizes. This means when population size is increased, the maxim number of iterations is reduced. Therefore, the optimizer may not fully converge before it consumes the computational resources completely.

Apart from the adverse consequence of increasing the population size, Figure 42 also reveals that the ranks of methods are strongly consistent when population size is changed. The UD and OD, for example, are almost always the worst methods whereas TNT and QBL are consistently among of the best techniques. Among all initialization methods, the performance of GLP fluctuates the most. Further studies may unveil more insights into the root cause of this unique behavior of GLP.

We also provide Table 45 to support the findings of Figure 42. The WTL stats in this table are produced in the same way as in Table 44. The Table 45 shows some alternative initialization techniques (*e.g.,* TNT, SBL, OBL, and QBL) considerably outperform RNG in most cases (*i.e.,* at least 83% of cases). This means that regardless of population size, RNG is far from the best choice for population initialization when dealing with large-scale optimization problems.

Another valuable observation from Table 45 and Figure 42 is that increasing population size cannot remedy RNG weakness. Increasing population size, indeed, not only adversely affect the functionality of RNG in all cases, it amplifies the gap between RNG and other studied initializers. For clear evidence, consider the number of times each method surpasses RNG when population size is 50 (see the first numbers of each WTL

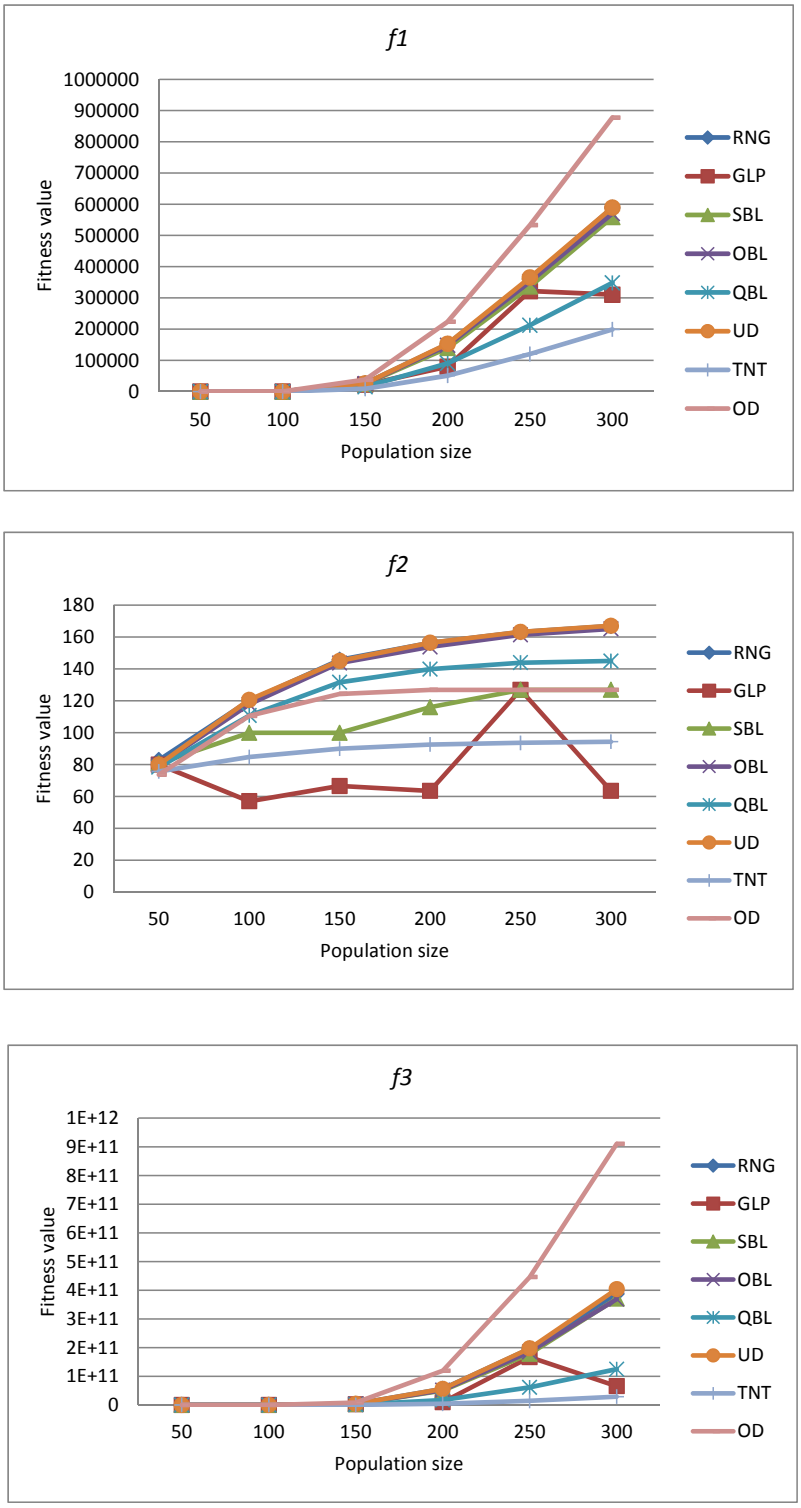


Figure 41: Comparison between advanced initialization methods with different population sizes on CEC'2008 LSGO benchmark functions(dimension size = 500)

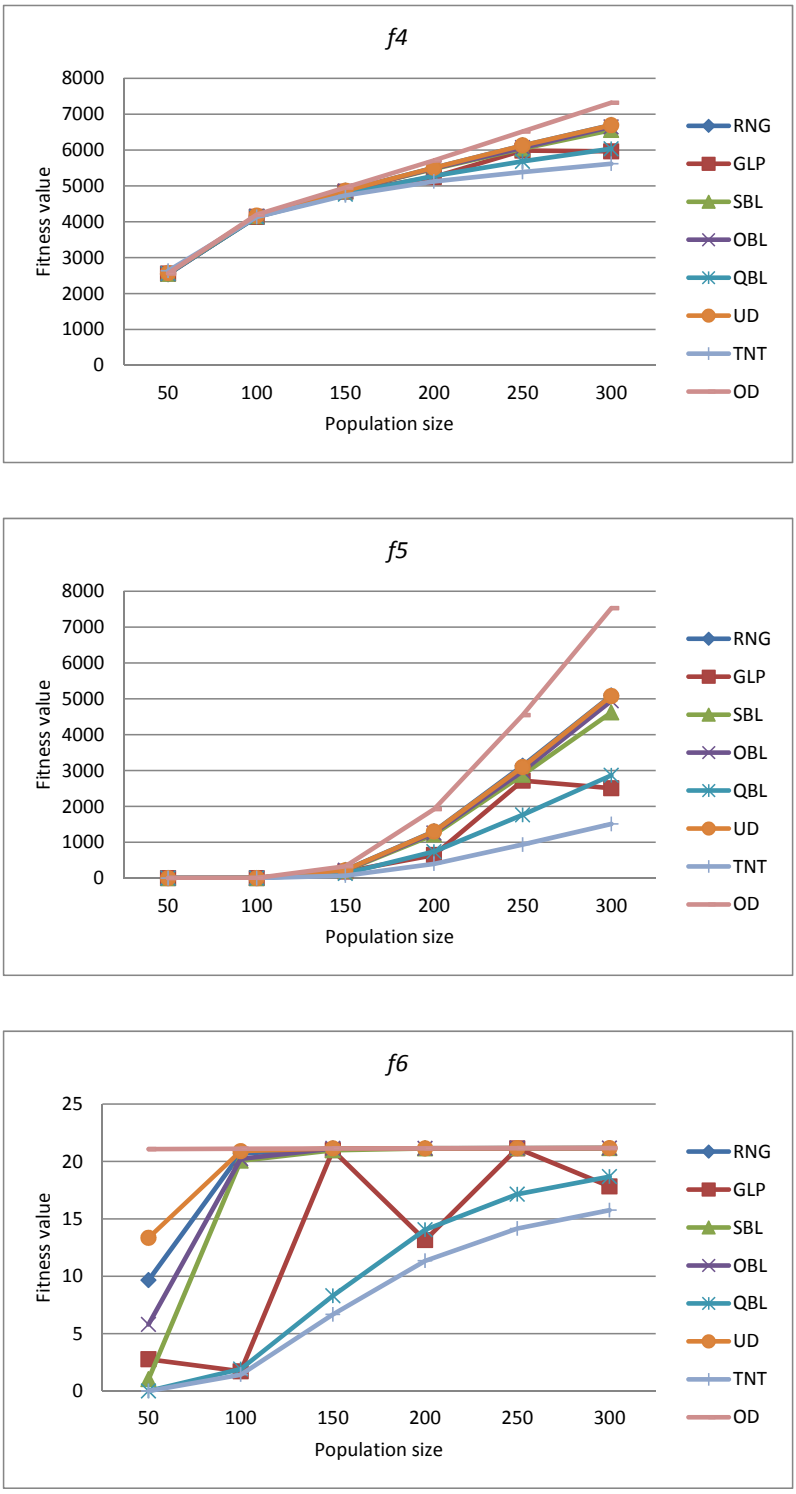


Figure 42: Comparison between advanced initialization methods with different population sizes on CEC'2008 LSGO benchmark functions(dimension size = 500)

triplet in the first row below the header of Table 45). Then, compare these values with the stats when population size is 300 (see the sixth row below the header of the table). It is obvious that RNG is beaten more times when the population size is 300 compared with the cases that population size is kept at 50.

From this part of the experiments, we conclude that RNG, even with a relatively large population, may not be the best choice for population initialization when dealing with large-scale problems. In contrast, the TNT, QBL, SBL, and GLP, can be considered as some better alternatives among the methods that we examined in this study.

Another observation from both parts of the experiment is that none of the initialization categories are significantly better than the others. While RNG, for example, does not perform well on large-scale problems, TNT which is from the same class as RNG, is ranked as one of the best alternatives. As another example compare OBL and QBL in Tables 44 and 45. Although these initializers look very similar in theory, they perform very differently in practice.

4.5 Chapter Summary

In this chapter, we studied the scalability of population initialization techniques to higher dimensions. In particular, we empirically examined several initializers that carefully selected from a wide range of techniques to investigate whether the algorithms that perform promisingly on small-sized problems can preserve their effectiveness on enormous optimization problems.

Our experiments revealed that dimensionality does not affect the rank of initializers in most cases. Moreover, we show that conventional pseudorandom number generator may not be the best option in the initialization step when the dimensionality is high, and computation resources are limited. However, in some cases, the gaps between the performance of algorithms expel as the dimensionality of functions grows. We also learned that different techniques from a similar class might perform significantly different in high-dimensional spaces. Finally, our analysis unveiled that increasing the size of the population when the overall budget kept fixed cannot enhance the effectiveness of the weak techniques.

In the next chapter, we will conduct more in-depth studies on the topic of population size and its effect on the functionality of population initializers when applied to large-scale optimization problems.

CHAPTER 5

Population Initialization and Optimizer Parameters

In Chapter 4, we empirically showed that by merely adopting a different population initialization technique one could save a substantial amount of computational budget and significantly improve the performance of optimizers in large-scale optimization tasks. In this chapter, we expand our studies to investigate whether other factors such as the size of the population can affect the effectiveness of the population initialization techniques. In other words, we intend to compare the significance of the effect of using alternative population initialization techniques with adopting the optimal parameter setting.

5.1 Motivation

The techniques based on Differential Evolution (DE) have always been among the best performers in the past optimization contests such as IEEE Congress on Evolutionary Computation (CEC) on large-scale global optimization problems [Qin and Li 2013]. Notwithstanding that, three key parameters can significantly affect the performance of DE: the population size, crossover rate, and mutation scaling factor. Therefore, several pieces of research studied the effect of these parameters and tried to identify their best values in different scenarios [Gamperle et al. 2002]. Some others proposed self-adaptive variants of DE that can reconfigure these parameters as the optimization progresses [Qin et al. 2009, Brest et al. 2006]. In addition to parameter tuning and adaptation, the quality of the initial population is also reported to have a significant impact on the effectiveness of DE variants. Hence, several population initialization techniques have been proposed and applied to DE [Rahnamayan et al. 2007a, Peng et al. 2010, Ali et al. 2012, de Melo and Botazzo Delbem 2012a].

Although the previous studies on DE parameter calibration and population initialization are scientifically valuable, they suffer from some shortcomings. Firstly, to the best of our knowledge, all works on DE parameter calibration are only done on low-dimensional search spaces. Therefore, the best parameter configuration for DE in dealing with large-scale problems remains to be discovered. So far, researchers and practitioners use the same values for solving low-dimensional and high-dimensional functions. To date, there is still no evidence to confirm or decline that the dimensionality of the tasks has any effect on the optimum configuration of DE.

Secondly, all the previous experiments on the DE initial population methods (including the analyses in Chapter 4) have been carried out using arbitrary parameter settings. Indeed, little attention has been paid to studying the effects of population initializers while the best parameter configuration is used. Obviously, the application of suboptimal parameter configuration can degrade the effectiveness of initial population. As a result, the derived conclusions from such studies may not be very precise or practical. This issue, which exists in studies on both low and high-dimensional problems, is the source of some contradictions and confusions on the effectiveness of advanced initializers [Morrison 2003].

To bridge these gaps in the literature, we conduct a systematic series of experiments to firstly identify the optimal parameter configuration for DE when dealing with large-scale optimization problems. More precisely, in this chapter, we investigate whether the best parameter values for low and high dimensions are equal.

Secondly, we compare the effects of some of the well-known population initializers that we reviewed in Chapter 3 and empirically studied in Chapter 4 in two scenarios: when the most commonly used parameter settings are adopted versus the best found parameter configuration from the previous part. In other words, this chapter investigates whether the quality of initial population has a significant impact on DE performance when different parameter settings are employed.

The answers to the questions asked above will shed more light on the mutual effects of DE parameters and initial population on its performance, especially when handling high-dimensional functions. Particularly, researchers and practitioners can use the provided advices for DE parameter calibration to solve their black-box optimization problems. Moreover, this chapter clarifies the relation between population initialization and DE main control parameters.

The rest of the chapter is as follows. The Section 5.2 presents a brief review of a widely adopted DE variants and its main control parameters. Then, we discuss the details of the conducted experiment in Section 5.3 and the results and findings in Section 5.4. Finally, Section 5.5 concludes the chapter.

5.2 Differential Evolution

The DE algorithm was originally proposed by Storn and Price and is one of the most effective and efficient stochastic optimization techniques [Storn and Price 1997]. After about two decades, DE has been developed into one of the most powerful and promising research topics in the field of evolutionary computation [Abbass 2002]. So far, a great and still growing body of literature is devoted to improving its performance [Chakraborty 2008], explaining its behavior [Zaharie 2002] and expanding its applications in numerous fields [Abbass 2002]. The family o1f DE variants has presented an exceptional performance when solving challenging optimization problems in different forms [Das and Suganthan 2011]. The DE-based techniques have been among the best performers in the past optimization competitions on a variety of optimization tasks such as single objective, multi-objective and large-scale global optimization problems [Qin and Li 2013].

DE is a *population-based* algorithm which means it starts with a set of candidate solutions (*i.e.,* initial population) which are usually drawn randomly from a uniform distribution within the solution space (see Chapter 3 for more details). Then, it applies its special operators on each individual in the population to produce new candidate solutions. The old and the new sets of candidate solutions are called the parent and child (or offspring) population, respectively. After that, DE evaluates both populations based on the given objective function (which is usually a black-box procedure) and substitutes

parent solution by its offspring if the offspring is a fitter solution than its parent. DE iteratively repeats the reproduction, evaluation, and selection steps until a termination criterion (*e.g.,* the maximum number of objective function evaluations) is met. Finally, it returns the fittest individual of the population as the best solution to the given optimization problem.

5.2.1 Differential Evolution Operators

In each iteration of DE process, we apply three operators to each single individual (*a.k.a. target vector*): *mutation*, *recombination* and *selection*. These operators work as follows:

Mutation: For each target vector, a *base* vector is chosen among the members of the current population and added to the scaled difference of a few randomly selected members. The resulting vector is called the *mutant* vector:

$$\mathbf{y}_i^t = \mathbf{x}_b^t + F \times (\mathbf{x}_{r_1}^t - \mathbf{x}_{r_2}^t),\ i = 1, 2, \ldots, \mathrm{N} \tag{5.1}$$

In Equation (5.1), the r_1 and r_2 are randomly chosen from $\{1, \ldots, N\}$ where N and t denote the population size and iteration number. The $\mathbf{x}_b$ and $\mathbf{y}_i$ are the base and mutant vectors, respectively. Note that various mutation strategies may follow different procedures to generate a mutant vector based on the target and base vectors.

Recombination: After the mutation step that we discussed in Equation (5.1), we can generate a *trial* vector combining the mutant and target vectors:

$$\mathbf{z}_{i,j}^t = \begin{cases} \mathbf{y}_{i,j}^t & \text{if } rand(0,1) \leq CR \text{ or } j = j_r, \\ \mathbf{x^t}_{i,j} & \text{otherwise.} \end{cases} \tag{5.2}$$

In Equation (5.2), $\mathbf{z}_i$ is the trial vector of the i^{th} target vector and j indicates the j^{th} variable of a vector. Here, the value of j_r is chosen randomly from $\{1, \ldots, N\}$ and whereas $rand(0, 1)$ generates a random number from the range $[0, 1]$. The CR in Equation (5.2) stands for *crossover rate* as disccussed earlier. There are several approaches for the recombination step and among them the discrete recombinations (*a.k.a.* crossovers) are the most widely used sachems.

Selection: Finally, DE compares each target vector (as the parent) and the corresponding trial (as its offspring) based on their fitness values. After the parent-child competition, the fittest candidate solution (*e.g.,* the one with the smallest objective value in a minimization task) remains in the population and the other individual usually dies.

$$\mathbf{x}_i^{t+1} = \begin{cases} \mathbf{z}_i^t & \text{if} f(\mathbf{x}_i^t) > f(\mathbf{z}_i^t), \\ \mathbf{x^t}_i & \text{otherwise.} \end{cases} \tag{5.3}$$

DE applies these three operators to all individuals in the population in a round-robin fashion. After this process completes, a DE generation is passed. Traditionally, DE repeats the same operations in the next generations until a termination criterion is met.

5.2.2 Differential Evolution Variants

DE has many variants which are usually denoted using DE/x/y/z convention, where:

- The *'x'* defines the base vector generation scheme. For example, 'x=best' indicates the current fittest member should be selected as the base vector whereas 'x=rand' means the base vector is selected randomly.

- The *'y'* defines the number of pairs of members that DE uses when constructing the difference vector(s) in Equation (5.1). The number of pairs is usually an integer value between one and three.
- The *'z'* defines the scheme of recombination. For example, 'z=bin' and 'z=exp' indicate the binomial (or uniform) and exponential (similar to circular two-point) crossovers, respectively.

5.2.3 Differential Evolution Parameters

The DE algorithm, in a very general form, has three main control parameters:

Population size (N): Like other population-based algorithm, N plays a crucial role in the efficiency and effectiveness of DE. Large population size potentially increases the population diversity and helps DE to sample more regions, simultaneously. However, when computational budget is limited (which in practice usually is), increasing the population size will decrease the number of iterations (*i.e.,* generations) and may result in early termination. In other words, DE may be terminated before the population converges to a desirable point.

Crossover rate (CR): In discrete recombination,CR value determines the number of decision variables of each target vector which must be interchanged with the corresponding variables of mutant vector. As a rule of thumb, small CR values can boost convergence speed when a few decision variables are interacting with each others. In turn, large CR values are more effective when lots of decision variables are interacting.

Mutation scale factor (F): In DE, the exploration-exploitation balance is controlled by F value. As a rule of thumb, too small F values increase the risk of premature convergence (*i.e.,* converge to an undesirable point), while too large F values decrease the convergence speed that degrades DE efficiency and may result in early termination.

Note that advanced variants of DE may have extra control parameters as they tend to have more complex components. So far, several strategies such as fixed [Price et al. 2005], control [Das et al. 2005] and adaptive [Qin and Suganthan 2005, Mallipeddi et al. 2011] schemes are proposed for DE parameter calibration. This work follows the framework suggested in [Qin and Li 2013] and hence, falls into the fixed scheme group.

5.3 Experiments

As mentioned earlier, we conduct two series of experiments in this chapter. In the first series of experiments, we find the most effective parameter configuration for a well-known DE model (see Section 5.2) when applied to large-scale optimization tasks. We are interested to study how dimensionality may affect the effectiveness of the DE parameters. In the second series of the experiments, we compare the effects of different population initialization techniques on the same DE model when the most common (according to literature) and the best (based on the findings from the first part) parameter configurations are used. The objective here is to investigate if different population initializers can make a statistically significant difference when the optimal parameter settings are in place.

In the both series of the experiments, we adopt the DE/rand/1/bin model because of its simplicity and popularity. This classical DE has been used in similar studies on DE parameter calibration in dealing with low dimensional problems [Qin and Li 2013]. We also use the CEC'13 benchmark set since it is the most recent and comprehensive benchmark suite available today for research on large-scale optimization. The succeed-

ing parts provide further information regarding the adopted benchmark tasks, conducted experimental setup, obtained results, and statistical analyses.

5.3.1 Benchmark Functions

The CEC'13 LSGO benchmark suite is currently the latest proposed benchmark in the field of large-scale optimization [Li et al. 2013a]. The suite consists of 15 continuous functions which are grouped into five distinct categories: fully separable functions ($f_1 - f_3$), partially separable functions with a separable component ($f_4 - f_7$), partially separable functions with no separable components ($f_8 - f_{11}$), overlapping functions ($f_{12} - f_{14}$) and one fully nonseparable function (f_{15}). All functions have 1000 decision variables, except f_{12} and f_{14} which have 905 variables due to overlapping components.

To improve previously proposed benchmark suite (*e.g.,* CEC'08 LSGO [Tang et al. 2007]), new functions with nonuniform component sizes and overlapping components have been added to the recent suite. Moreover, new transformations such as ill-conditioning, symmetry breaking and irregularities have been added to CEC'13 LSGO suite. More details regarding this suite are available in [Li et al. 2013a].

5.3.2 Experiments Setup

This section consists of two major parts: the DE parameter calibration for large-scale problem and the study of the performance of alternative population initializers. The following paragraphs discuss the setup of all parts in details.

A. Parameter Calibration

In the first series of experiments, we evaluate the performance of DE/rand/1/bin on all 15 functions of CEC'13 LSGO benchmark suite using 84 different parameter configurations. These parameter configurations consist of all possible combinations of 14 population sizes (*i.e.,* $N \in [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200, 250, 300]$), three commonly used crossover rates (*i.e.,* $CR \in [0.1, 0.5, 0.9]$), and two widely adopted mutation scaling factors (*i.e.,* $F \in [0.5, 0.8]$). These configurations cover most of the advised values in the previously published studies [Qin and Li 2013].

For each benchmark function, we execute the DE/rand/1/bin algorithm with all 84 configurations for 51 independent trials. Following the framework suggested in [Qin and Li 2013], the i^{th} runs of all configurations initialized by the same seed while i^{th} and j^{th} ($i \neq j$) trials of any configuration differ in initial seed. To be consistent with CEC'13 LSGO competition rules, we restrict the maximum number of function evaluations to $3 \times D = 3,000,000$ for all experiments.

B. Population Initialization

In the second series of experiments, we evaluate the effects of six population initialization techniques on the performance of DE/rand/1/bin on all 15 functions of CEC'13 LSGO benchmark suite. Besides the common random population initializer (*i.e.,* RNG), five well-known and potentially effective initializers are also employed.

For this part, we select the Mersenne Twister [Matsumoto and Nishimura 1998] and tent map [Dong et al. 2012] from RNG and TNT groups, respectively. These techniques are stochastic population generators which each execution of them with different initial seeds will result in a different set of initial points. In contrast, we chose the Sobol set (SBL) [Bratley and Fox 1988] and Good Lattice Points [Sloan and Joe 1994] form the

deterministic population generator category. As per the literature, we expect them to produce evenly scattered points with a high level of uniformity. To complement these four techniques, we adopt the Opposition-Based Learning (OBL) [Rahnamayan et al. 2007a] and Quasi-opposition-Based Learning (QBL) [Rahnamayan et al. 2007c] initializers from the greedy algorithms. These methods, generate a set of candidate solutions from the seeded population and then evaluate all solutions to select the optimal subset (in terms of fitness value) as the optimizer's initial population. In all cases, we adopt the common values for the control parameters of the initialization techniques as discussed in Chapter 4

In this part of the experiments, we study two different sets of parameter configurations for DE/rand/1/bin: the most common parameter configuration we found in the literature and the most effective configuration found from the first part of the experiments (*i.e.,* parameter calibration).

We run the DE/rand/1/bin with six population initialization techniques and two configurations to solve each of the minimization problems for 51 times. Similar to the first part, the i^{th} trials of all six initializers use the same initial seed whereas the initial seeds for trials i^{th} and j^{th} $(i \neq j)$ of any technique are chosen differently from a random set. Following the same procedure as the first part, we limit the maximum number of objective function calls to $3e{+}06$ per trial.

5.4 Results and Discussion

We dedicate the following parts to the analysis and discussions of the obtained experimental results.

A. Parameter Calibration

We employ a number of advanced nonparametric statistical tools to systematically find the most effective parameter configuration(s) among 84 different predefined configurations of DE/rand/1/bin on CEC'13 LSGO benchmark. We use Iman and Davenport test as a variant of Friedman rank test to rank the configurations [Derrac et al. 2011]. According to the obtained this ranking, $[N,\ CR,\ F]{=}[150, 0.9, 0.5]$ is the most effective parameter setting[1] for DE/rand/1/bin on these particular benchmark problems. Table 51 reports the summary statistics of the results obtained using this configuration. For the sake of comparison, we also include the results obtained by using the most common configuration ($[N,\ CR,\ F]{=}[50, 0.9, 0.5]$), which is also previously reported as the most effective configuration on low dimensional problems [Qin and Li 2013]. The Table 51 clearly shows that the performance of DE/rand/1/bin in most cases is significantly improved when the optimal configuration is used.

Note that the ranking is calculated in respect of the configurations performance on all 15 functions as we do not compare configurations performance on every single function separately. Two reasons support avoiding such statistical analysis: Firstly, according to [Derrac et al. 2011], at least 252 independent trials of each function (per configuration) are needed to compare 84 algorithms. Otherwise, the comparison may not be statistically meaningful. Secondly, we aim to provide some general rules of thumb for practitioners to tune the parameters of DE/rand/1/bin in dealing with large-scale black-box problems. Therefore, the general assumption is that practitioners have no extensive prior knowledge about the problems at hand. Consequently, even if we report the most efficient configu-

[1] We carefully use the term optimal configuration with regard to the 84 examined options. There is a slight chance that global optimum values for the parameters exist outside of the studied ranges.

ration for every single function of the benchmark suite, they may not be able to find the proper settings for their particular application.

We must apply some post-hoc procedures to determine which configurations are significantly dominated by the optimal setting. In this study, we employ the Li post-hoc procedure [García et al. 2010]. According to [Derrac et al. 2011], the Li post-hoc procedure is one of the most robust procedures among the standard statistical tools. Generally, post-hoc methods compare the distribution of results from all methods against the outcome of the control method to investigate whether they perform significantly different.

In this part of the experiment, we treat each parameter configuration as a single algorithm and adopt the optimal configuration which is identified by Iman and Davenport ranking procedure as the control method. Based on the Li's adjusted p-values, the configurations are divided into two groups: those which their performance significantly worse than the control method, and those which perform statistically similar to the control configuration. Table 52 demonstrate the results in details.

As Table 52 indicates, the optimal values for CR and F are 0.9 and 0.5, respectively. These values are consistent with the findings from [Qin and Li 2013] for low dimensional problems (*i.e.,* $D \in [10, 30, 50]$) where the same values are identified as the most effective configurations for DE/rand/1/bin. Table 52 also reveals that the most effective range of population size for solving high-dimensional problems with regard to the dedicated computational budget is $80 \leq N \leq 250$. In comparison with the previous findings on low-dimensional problems (*i.e.,* $40 \leq N \leq 60$), our results show that DE/rand/1/bin (and most certainly other population-based metaheuristics) needs larger population sizes as the dimensionality of search space increases. In other words, while the effective CR and F values for both low and high-dimensional problems are the same, optimal population size depends on the number of decision variables.

It should be noted that these and previous findings are based on the dedicated computational budget (*i.e.,* $3e{+}06$ maximum number of function evaluations). A very different computational budget may affect the outcomes. As an intuitive rule of thumb, providing a larger pool of resources allows the optimizer to leverage larger populations more effectively.

A direct comparison between the obtained results from this study and the experiments on low dimensional problems from [Qin and Li 2013] is impossible. The reason is these studies adopt significantly different benchmarks and the computational budgets. Therefore, instead of a statistical comparison, we only compare our findings with the previously published insights.

Table 51: Comparison between the obtained results from the most common ([N, CR, F]=[50, 0.9, 0.5]) and the most effective ([N, CR, F]=[150, 0.9, 0.5]) configurations for DE/rand/1/bin on CEC'13 LSGO benchmarks. C_C and C_S stand for the most Common and the Superior configurations, respectively. $b, \tilde{m}, w, \mu$, and σ stand for the best, median, worst, mean, and standard deviation of 51 independent runs, respectively.

Conf.	Stat.	Group 1			Group 2				Group 3				Group 4			Group 5
		f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
$\mathbf{C_C}$	b	5.0e+05	1.9e+04	2.0e+01	1.7e+10	1.5e+06	1.1e+06	5.1e+07	8.9e+13	1.4e+08	9.3e+07	3.9e+10	4.5e+08	4.6e+09	9.6e+10	6.1e+07
	$\tilde{m}$	4.2e+06	2.0e+04	2.1e+01	4.3e+10	2.2e+06	1.1e+06	1.2e+08	3.5e+14	2.1e+08	9.4e+07	1.5e+11	5.1e+09	8.6e+09	2.0e+11	6.8e+09
	w	3.9e+08	2.2e+04	2.1e+01	9.0e+10	3.2e+06	1.1e+06	8.5e+08	7.9e+14	2.8e+08	9.5e+07	6.4e+11	2.0e+10	1.5e+10	4.3e+11	6.7e+11
	μ	3.3e+07	2.0e+04	2.1e+01	4.6e+10	2.3e+06	1.1e+06	1.5e+08	3.7e+14	2.1e+08	9.4e+07	1.7e+11	6.0e+09	8.9e+09	2.1e+11	5.2e+10
	σ	8.0e+07	9.2e+02	8.9e-02	1.7e+10	4.1e+05	1.1e+03	1.2e+08	1.7e+14	3.0e+07	2.5e+05	1.1e+11	4.9e+09	2.1e+09	8.4e+10	1.2e+11
$\mathbf{C_S}$	b	6.4e+05	7.2e+03	1.1e+01	5.5e+09	7.5e+06	1.8e+01	9.3e+07	2.7e+12	5.4e+07	6.7e+01	5.5e+09	3.0e+07	3.6e+09	4.2e+10	2.8e+07
	$\tilde{m}$	1.7e+06	8.4e+03	1.3e+01	1.5e+10	8.4e+06	2.0e+01	2.1e+08	1.5e+13	1.4e+08	1.5e+02	4.9e+10	8.3e+07	5.3e+09	8.2e+10	5.5e+07
	w	1.0e+07	9.5e+03	1.4e+01	3.0e+10	8.7e+06	3.8e+01	4.5e+08	1.0e+14	6.5e+08	3.3e+02	1.1e+11	3.8e+09	9.4e+09	1.9e+11	1.2e+08
	μ	2.0e+06	8.4e+03	1.3e+01	1.5e+10	8.4e+06	2.0e+01	2.3e+08	2.2e+13	2.7e+08	1.6e+02	5.0e+10	1.8e+08	5.5e+09	8.4e+10	6.1e+07
	σ	1.5e+06	5.1e+02	7.1e-01	5.5e+09	2.7e+05	2.9e+00	9.0e+07	1.8e+13	2.3e+08	5.4e+01	2.3e+10	5.3e+08	1.3e+09	2.7e+10	2.0e+07

Table 52: The optimal parameter configurations according to Iman and Davenport test with Li post-hoc procedure on all 15 functions of CEC'13 LSGO benchmark suite. Among 84 examined configurations, those leading to the statistically better performance with 0.05 significance level over others are marked by ✓. The others are statistically worse than the marked configurations.

F	CR	N													
		10	20	30	40	50	60	70	80	90	100	150	200	250	300
0.5	0.1	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	0.5	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	0.9	-	-	-	-	-	-	-	✓	✓	✓	✓	✓	✓	-
0.8	0.1	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	0.5	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	0.9	-	-	-	-	-	-	-	-	-	-	-	-	-	-

B. Population Initialization

To study the effects of alternative population initialization techniques on large-scale problems, we conduct a study to compare six promising initializers with two configurations. We report the obtained results from DE/rand/1/bin with the most effective ([N, CR, F]=[150, 0.9, 0.5]) and the most common ([N, CR, F]=[50, 0.9, 0.5]) configurations using the six population initialization techniques in Tables 54 and 53, respectively. As these tables present, the alternative initializers improved the common RNG algorithm's outcome in some functions for both configurations.

We apply Iman and Davenport with the Li post-hoc procedure to identify the most effective initialization technique(s) among the studies algorithms. However, due to the large values of the calculated p-values (*i.e.,* greater than 0.05), the obtained ranks are not statistically meaningful when the most effective configuration is used. In other words, although some improvements are achieved by employing alternative population initialization techniques, the improvements are not statistically significant from a statistical point of view. This means all population initialization techniques perform statistically similar when they are applied to DE/rand/1/bin with well-tuned control parameters. This is a new finding that indicates when proper values for the control parameters are used, population initialization has only a minor effect on the optimizer performance. When the computational budget is capped at $3e{+}06$ function evaluations, increasing population size from 50 to 150 has a more significant impact than changing the initializer algorithm.

This new finding is very valuable because it challenges the common belief in the effect of population initialization techniques in the improvement of population-based metaheuristics on large-scale problems. Although some previous works (*e.g.,* [Morrison 2003, Morokoff and Caflisch 1994, Wang and Sloan 2008]) raise doubt about the effectiveness of some techniques in high-dimensional spaces, the common belief was that alternative methods could improve the metaheuristics performance regardless of dimensionality.

The contradiction between the new findings and some of the previous claims have root in two shortcomings of some of the earlier studies. Firstly, to the best of our knowledge, none of these studies has tried to compare population initialization techniques on the well-tuned optimizers. Neglecting the significant effects of main control parameters, especially the population size, on the performance of the optimizers may result in a biased conclusion.

Secondly, the necessity of employing advanced statistical tools such as those recommended in [Derrac et al. 2011] and used in this study for validating the findings is neglected in most of the previous work. Consequently, some statistically minor improvements as a result of using alternative initializers may incorrectly be considered as significant contributions.

Note that this study is conducted based on a systematic framework and the findings are statistically validated. However, we are well aware of the need for further investigations to generalize the results from DE/rand/1/bin to other optimization algorithms.

Table 53: The performance of six population initialization techniques using the common parameter configuration ([N, CR, F]=[50, 0.9, 0.5]) for DE/rand/1/bin. RNG, TNT, SBL, GLP, OBL and QBL stand for pseudo-random number generator, chaotic number generator, Sobol set, good lattice points, opposition-based learning and quasi-opposition-based learning, respectively. $b, \tilde{m}, w, \mu$, and σ stand for the best, median, worst, mean, and standard deviation of 51 runs, respectively.

Init.	**Stat.**	**Group 1**			**Group 2**				**Group 3**				**Group 4**			**Group 5**
		f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
RNG	b	5.0e+05	1.9e+04	2.0e+01	1.7e+10	1.5e+06	1.1e+06	5.1e+07	8.9e+13	1.4e+08	9.3e+07	3.9e+10	4.5e+08	4.6e+09	9.6e+10	6.1e+07
	$\tilde{m}$	4.2e+06	2.0e+04	2.1e+01	4.3e+10	2.2e+06	1.1e+06	1.2e+08	3.5e+14	2.1e+08	9.4e+07	1.5e+11	5.1e+09	8.6e+09	2.0e+11	6.8e+09
	w	3.9e+08	2.2e+04	2.1e+01	9.0e+10	3.2e+06	1.1e+06	8.5e+08	7.9e+14	2.8e+08	9.5e+07	6.4e+11	2.0e+10	1.5e+10	4.3e+11	6.7e+11
	μ	3.3e+07	2.0e+04	2.1e+01	4.6e+10	2.3e+06	1.1e+06	1.5e+08	3.7e+14	2.1e+08	9.4e+07	1.7e+11	6.0e+09	8.9e+09	2.1e+11	5.2e+10
	σ	8.0e+07	9.2e+02	8.9e-02	1.7e+10	4.1e+05	1.1e+03	1.2e+08	1.7e+14	3.0e+07	2.5e+05	1.1e+11	4.9e+09	2.1e+09	8.4e+10	1.2e+11
TNT	b	7.0e+05	1.7e+04	2.1e+01	2.2e+10	1.5e+06	1.1e+06	5.4e+07	9.8e+13	1.5e+08	9.3e+07	4.2e+10	3.7e+08	4.2e+09	9.7e+10	1.7e+08
	$\tilde{m}$	5.6e+06	2.0e+04	2.1e+01	4.1e+10	2.2e+06	1.1e+06	1.3e+08	3.4e+14	2.0e+08	9.4e+07	1.5e+11	6.0e+09	8.3e+09	2.0e+11	1.1e+10
	w	2.6e+08	2.3e+04	2.1e+01	7.4e+10	3.1e+06	1.1e+06	4.5e+08	7.8e+14	2.6e+08	9.4e+07	4.6e+11	1.8e+10	1.2e+10	4.9e+11	6.3e+11
	μ	2.1e+07	2.0e+04	2.1e+01	4.4e+10	2.2e+06	1.1e+06	1.5e+08	3.4e+14	2.0e+08	9.4e+07	1.8e+11	6.4e+09	8.1e+09	2.2e+11	5.2e+10
	σ	4.2e+07	9.7e+02	8.2e-02	1.4e+10	4.0e+05	1.1e+03	8.0e+07	1.7e+14	2.8e+07	2.7e+05	1.1e+11	4.2e+09	1.9e+09	8.6e+10	1.1e+11
SBL	b	8.8e+05	1.8e+04	2.0e+01	1.6e+10	1.5e+06	1.1e+06	4.6e+07	1.0e+14	1.4e+08	9.3e+07	6.7e+10	3.9e+08	5.3e+09	8.2e+10	6.6e+06
	$\tilde{m}$	7.7e+06	2.0e+04	2.1e+01	4.4e+10	2.2e+06	1.1e+06	1.4e+08	3.4e+14	2.2e+08	9.4e+07	1.8e+11	4.2e+09	8.7e+09	2.1e+11	1.0e+07
	w	2.2e+08	2.3e+04	2.1e+01	9.4e+10	3.3e+06	1.1e+06	4.5e+08	6.0e+14	2.8e+08	9.5e+07	4.3e+11	1.1e+10	1.5e+10	4.4e+11	1.5e+07
	μ	2.1e+07	2.0e+04	2.1e+01	4.6e+10	2.3e+06	1.1e+06	1.8e+08	3.3e+14	2.1e+08	9.4e+07	2.0e+11	4.8e+09	9.2e+09	2.2e+11	1.0e+07
	σ	3.7e+07	1.0e+03	9.9e-02	1.9e+10	4.1e+05	9.4e+02	1.2e+08	1.2e+14	3.4e+07	2.4e+05	9.3e+10	3.2e+09	2.3e+09	8.0e+10	2.0e+06
GLP	b	1.0e+06	1.8e+04	2.0e+01	1.8e+10	1.2e+06	1.1e+06	6.2e+07	8.5e+13	1.5e+08	9.4e+07	1.5e+10	1.6e+08	4.0e+09	7.7e+10	7.0e+06
	$\tilde{m}$	1.1e+07	2.0e+04	2.1e+01	4.5e+10	2.3e+06	1.1e+06	1.3e+08	3.4e+14	2.0e+08	9.4e+07	1.3e+11	4.7e+09	7.5e+09	2.2e+11	9.7e+06
	w	2.0e+08	2.2e+04	2.1e+01	1.2e+11	3.4e+06	1.1e+06	5.1e+08	7.6e+14	3.0e+08	9.5e+07	6.7e+11	1.8e+10	1.5e+10	3.7e+11	2.0e+07
	μ	3.6e+07	2.0e+04	2.1e+01	4.7e+10	2.3e+06	1.1e+06	1.6e+08	3.4e+14	2.1e+08	9.4e+07	1.6e+11	6.0e+09	7.9e+09	2.2e+11	1.0e+07
	σ	5.1e+07	8.0e+02	1.0e-01	2.0e+10	4.8e+05	1.2e+03	8.5e+07	1.4e+14	3.0e+07	2.2e+05	1.0e+11	4.7e+09	2.4e+09	8.2e+10	2.6e+06
OBL	b	8.9e+05	1.8e+04	2.0e+01	1.3e+10	1.2e+06	1.1e+06	4.1e+07	1.1e+14	1.5e+08	9.3e+07	3.6e+10	6.0e+08	3.8e+09	9.9e+10	1.5e+08
	$\tilde{m}$	4.8e+06	2.0e+04	2.1e+01	4.4e+10	2.3e+06	1.1e+06	1.3e+08	3.0e+14	2.1e+08	9.4e+07	1.6e+11	4.2e+09	8.2e+09	1.9e+11	9.8e+09
	w	9.1e+08	2.3e+04	2.1e+01	8.9e+10	3.3e+06	1.1e+06	7.6e+08	7.4e+14	3.1e+08	9.4e+07	4.5e+11	1.5e+10	1.3e+10	5.1e+11	4.4e+11
	μ	5.5e+07	2.0e+04	2.1e+01	4.6e+10	2.3e+06	1.1e+06	1.7e+08	3.3e+14	2.1e+08	9.4e+07	1.7e+11	5.2e+09	8.2e+09	2.2e+11	3.6e+10
	σ	4.6e+07	8.0e+02	1.0e-01	1.7e+10	4.1e+05	1.0e+03	1.0e+08	1.4e+14	3.5e+07	2.2e+05	1.4e+11	4.1e+09	1.9e+09	9.0e+10	2.5e+11
QBL	b	6.8e+05	1.9e+04	2.0e+01	1.9e+10	1.4e+06	1.1e+06	5.3e+07	6.5e+13	1.7e+08	9.3e+07	3.6e+10	5.0e+08	4.5e+09	1.1e+11	3.9e+08
	$\tilde{m}$	4.0e+06	2.1e+04	2.1e+01	4.7e+10	2.2e+06	1.1e+06	1.4e+08	3.3e+14	2.2e+08	9.4e+07	1.8e+11	4.4e+09	7.8e+09	1.8e+11	2.1e+10
	w	2.6e+08	2.2e+04	2.1e+01	9.0e+10	3.4e+06	1.1e+06	5.4e+08	6.5e+14	3.0e+08	9.4e+07	7.3e+11	1.8e+10	1.3e+10	5.5e+11	1.7e+12
	μ	1.6e+07	2.0e+04	2.1e+01	4.7e+10	2.3e+06	1.1e+06	1.7e+08	3.4e+14	2.2e+08	9.4e+07	2.2e+11	5.4e+09	8.0e+09	2.0e+11	8.9e+10
	σ	4.6e+07	8.0e+02	1.0e-01	1.7e+10	4.1e+05	1.0e+03	1.0e+08	1.4e+14	3.5e+07	2.2e+05	1.4e+11	4.1e+09	1.9e+09	9.0e+10	2.5e+11

Table 54: The performance of six population initialization techniques using the most effective parameter configuration ([N, CR, F]=[150, 0.9, 0.5]) for DE/rand/1/bin. RNG, TNT, SBL, GLP, OBL and QBL stand for pseudo-random number generator, chaotic number generator, Sobol set, good lattice points, opposition-based learning and quasi-opposition-based learning, respectively. $b, \tilde{m}, w, \mu$, and σ stand for the best, median, worst, mean, and standard deviation of 51 runs, respectively.

Init.	**Stat.**	**Group 1**			**Group 2**				**Group 3**				**Group 4**			**Group 5**
		f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
RNG	b	6.4e+05	7.2e+03	1.1e+01	5.5e+09	7.5e+06	1.8e+01	9.3e+07	2.7e+12	5.4e+07	6.7e+01	5.5e+09	3.0e+07	3.6e+09	4.2e+10	2.8e+07
	$\tilde{m}$	1.7e+06	8.4e+03	1.3e+01	1.5e+10	8.4e+06	2.0e+01	2.1e+08	1.5e+13	1.4e+08	1.5e+02	4.9e+10	8.3e+07	5.3e+09	8.2e+10	5.5e+07
	w	1.0e+07	9.5e+03	1.4e+01	3.0e+10	8.7e+06	3.8e+01	4.5e+08	1.0e+14	6.5e+08	3.3e+02	1.1e+11	3.8e+09	9.4e+09	1.9e+11	1.2e+08
	μ	2.0e+06	8.4e+03	1.3e+01	1.5e+10	8.4e+06	2.0e+01	2.3e+08	2.2e+13	2.7e+08	1.6e+02	5.0e+10	1.8e+08	5.5e+09	8.4e+10	6.1e+07
	σ	1.5e+06	5.1e+02	7.1e-01	5.5e+09	2.7e+05	2.9e+00	9.0e+07	1.8e+13	2.3e+08	5.4e+01	2.3e+10	5.3e+08	1.3e+09	2.7e+10	2.0e+07
TNT	b	5.1e+05	7.3e+03	1.1e+01	4.7e+09	7.7e+06	1.8e+01	1.1e+08	3.1e+12	7.3e+07	7.5e+01	1.2e+10	1.6e+07	3.3e+09	4.3e+10	3.4e+07
	$\tilde{m}$	1.7e+06	8.3e+03	1.3e+01	1.5e+10	8.3e+06	2.0e+01	2.2e+08	2.2e+13	1.4e+08	1.5e+02	4.7e+10	8.6e+07	5.7e+09	8.5e+10	6.3e+07
	w	2.2e+07	9.4e+03	1.4e+01	2.8e+10	8.9e+06	3.2e+01	7.4e+08	8.5e+13	6.7e+08	5.4e+02	1.1e+11	7.1e+08	9.0e+09	1.7e+11	1.5e+08
	μ	2.7e+06	8.3e+03	1.3e+01	1.4e+10	8.3e+06	2.1e+01	2.3e+08	2.6e+13	3.2e+08	1.6e+02	5.0e+10	1.3e+08	5.8e+09	8.7e+10	6.7e+07
	σ	3.9e+06	4.5e+02	6.3e-01	5.2e+09	2.7e+05	2.8e+00	1.1e+08	2.0e+13	2.4e+08	8.4e+01	2.1e+10	1.3e+08	1.2e+09	2.7e+10	2.3e+07
SBL	b	6.6e+05	7.2e+03	1.2e+01	4.8e+09	7.5e+06	1.8e+01	1.2e+08	7.8e+11	7.0e+07	7.6e+01	1.0e+10	2.4e+07	3.0e+09	3.3e+10	4.3e+07
	$\tilde{m}$	1.9e+06	8.5e+03	1.3e+01	1.4e+10	8.4e+06	2.0e+01	2.3e+08	1.9e+13	1.5e+08	1.4e+02	5.3e+10	9.4e+07	5.6e+09	8.4e+10	7.9e+07
	w	1.4e+07	9.5e+03	1.4e+01	2.7e+10	8.9e+06	2.7e+01	6.1e+08	1.0e+14	6.6e+08	4.0e+02	1.3e+11	3.4e+08	7.9e+09	1.5e+11	1.3e+08
	μ	2.6e+06	8.5e+03	1.3e+01	1.5e+10	8.3e+06	2.0e+01	2.6e+08	2.5e+13	3.5e+08	1.6e+02	5.8e+10	1.2e+08	5.6e+09	8.8e+10	7.9e+07
	σ	2.3e+06	4.8e+02	5.8e-01	5.8e+09	2.9e+05	1.5e+00	1.1e+08	2.1e+13	2.5e+08	6.0e+01	2.6e+10	8.0e+07	1.1e+09	2.5e+10	1.9e+07
GLP	b	5.5e+05	7.6e+03	1.2e+01	6.2e+09	7.4e+06	1.8e+01	1.1e+08	4.0e+12	6.1e+07	7.4e+01	1.0e+10	2.2e+07	3.6e+09	4.3e+10	7.8e+07
	$\tilde{m}$	1.6e+06	8.6e+03	1.3e+01	1.5e+10	8.3e+06	2.1e+01	2.1e+08	1.4e+13	1.4e+08	1.5e+02	4.7e+10	9.5e+07	5.9e+09	8.1e+10	1.4e+08
	w	2.0e+07	9.4e+03	1.5e+01	2.9e+10	8.8e+06	3.0e+01	7.1e+08	9.4e+13	6.6e+08	3.0e+02	1.6e+11	1.1e+09	7.9e+09	1.5e+11	1.9e+08
	μ	2.6e+06	8.6e+03	1.3e+01	1.5e+10	8.3e+06	2.1e+01	2.4e+08	2.1e+13	3.2e+08	1.6e+02	5.4e+10	1.6e+08	5.9e+09	8.4e+10	1.4e+08
	σ	3.2e+06	4.4e+02	6.8e-01	5.6e+09	3.0e+05	1.8e+00	1.2e+08	1.7e+13	2.4e+08	5.5e+01	2.9e+10	1.8e+08	1.0e+09	2.4e+10	2.2e+07
OBL	b	5.8e+05	7.2e+03	1.1e+01	6.5e+09	7.7e+06	1.7e+01	9.6e+07	1.1e+12	7.2e+07	7.6e+01	1.1e+10	2.5e+07	2.9e+09	3.1e+10	3.7e+07
	$\tilde{m}$	2.0e+06	8.4e+03	1.3e+01	1.5e+10	8.3e+06	2.0e+01	2.2e+08	1.8e+13	1.4e+08	1.6e+02	4.5e+10	7.5e+07	6.0e+09	7.8e+10	6.5e+07
	w	1.1e+07	9.6e+03	1.4e+01	3.5e+10	8.8e+06	2.2e+01	4.1e+08	9.9e+13	6.6e+08	3.3e+02	1.0e+11	9.5e+08	8.4e+09	1.7e+11	1.7e+08
	μ	2.3e+06	8.4e+03	1.3e+01	1.6e+10	8.3e+06	2.0e+01	2.2e+08	2.3e+13	3.3e+08	1.7e+02	5.0e+10	1.4e+08	5.8e+09	8.1e+10	7.1e+07
	σ	1.9e+06	4.7e+02	6.7e-01	5.8e+09	2.6e+05	1.3e+00	8.3e+07	2.1e+13	2.4e+08	6.5e+01	2.2e+10	1.8e+08	1.2e+09	2.6e+10	2.4e+07
QBL	b	4.5e+05	8.0e+03	1.1e+01	4.3e+09	7.5e+06	1.7e+01	1.1e+08	2.1e+12	6.3e+07	7.1e+01	1.9e+10	2.1e+07	3.5e+09	3.6e+10	3.6e+07
	$\tilde{m}$	1.6e+06	8.6e+03	1.3e+01	1.3e+10	8.3e+06	2.0e+01	2.3e+08	1.7e+13	1.5e+08	1.6e+02	5.1e+10	8.1e+07	6.0e+09	8.2e+10	6.5e+07
	w	1.6e+07	9.7e+03	1.5e+01	3.3e+10	8.8e+06	3.5e+01	5.3e+08	5.3e+13	6.7e+08	5.6e+02	1.0e+11	9.2e+08	1.1e+10	1.3e+11	1.9e+08
	μ	2.5e+06	8.7e+03	1.3e+01	1.5e+10	8.3e+06	2.0e+01	2.4e+08	2.0e+13	3.3e+08	1.6e+02	5.3e+10	1.2e+08	5.9e+09	8.1e+10	7.5e+07
	σ	2.7e+06	4.5e+02	6.4e-01	6.4e+09	2.6e+05	2.5e+00	1.0e+08	1.3e+13	2.5e+08	7.2e+01	2.3e+10	1.3e+08	1.3e+09	2.3e+10	2.9e+07

5.5 Chapter Summary

In this chapter, we studied the effect of population initialization techniques in two different scenarios: when the common parameter configuration is used and when the parameter values are tuned. Our empirical analysis showed that in contrast with the common scenario, the alternative techniques can only marginally improve the functionality of the metaheuristics in dealing with large-scale optimization problems. We practically showed that the followings will lead to statistically similar results: adopting a promising alternative initializer in cases that the optimal population size is unknown and tuning the population size while continuing the use of traditional pseudo-random point generators.

These findings suggest that the gap between the common pseudo-random number generators and more advanced population initializers reduces when the population size is set to the optimal value. Note that the optimal population size is constrained by the computational budget, features of the search landscape and power of the optimizer. What is still remained unknown is that what will happen if there are no such constraints? Can we identify a piece of general advice on the adoption of different initializers regardless of the problem and optimizers features? In Chapter 6 we conduct a series of experiments to address these research questions.

CHAPTER 6

Uniformity and Curse of Dimensionality

In the two previous chapters, we show that starting from a better initial population can save a considerable portion of the limited computational budget. Therefore, the optimizer can continue the search process further and potentially achieve significantly better results. However, we also observed that as the dimensionality of the problem grows, the benefit of using some of these initializers dramatically drops. In this chapter, we study the uniformity of populations generated by different techniques as a proxy of the quality of the initial population. We are interested in investigating whether alternative methods can maintain the uniformity of the generated population in the higher dimensions. This can explain the performance gain or loss of using alternative approaches in solving high-dimensional problems.

6.1 Motivation

In Chapter 4, we studied the scalability of several alternative initialization techniques to see whether they can significantly enhance the performance of optimizers when the dimensionality of a problem is beyond a hundred variables. We showed that even in high-dimensional spaces some population initialization techniques improve the metaheuristics efficiency by boosting the convergence process and returning fitter solutions. However, we observed that the effectiveness of the studied initializers degrades as the dimensionality increases from 100 to 1,000. The experimental results also suggest that some of the techniques perform significantly worse than conventional pseudo-random number generators (RNGs). Furthermore, the studies carried out in Chapter 5 indicate that when the optimal population size is used, the improvement gained from alternative initializers becomes very marginal. Based on this observation, we concluded that the size of the initial population is as crucial as the algorithm we adopt to generate it.

Now, two questions remain unanswered:

1. Whether the previous findings can be generalized to other metaheuristics and optimization problems?

2. Why do alternative initializers not lead to significantly better optimization algorithm performance in high-dimensional spaces?

In this chapter, we answer the research questions mentioned above. The challenge here is that studying the effect of each available initializer on all metaheuristic algorithms

using many large-scale problems is practically impossible. Therefore, we narrow down the domain of our research to some techniques that are specially designed to generate uniform populations. Assuming uniformity as a critical factor in the performance of this group of initializers, investigating their scalability via the available uniformity measures is more practical than evaluating the performance of a large group of optimization algorithms on a variety of large-scale problems. As a result, we adopt general purpose tools to measure the uniformity of populations generated by different techniques to study the effect of dimensionality on their performance. The employed measures are carefully selected to guarantee the generality of the findings regardless of the type of optimization algorithm or problem.

6.2 Uniformity Measures

6.2.1 Background

Without a set of general and practical measures, we cannot accurately assess or compare the population initialization methods. Therefore, many different tools have been proposed to measure the quality of a set of points from various aspects, such as uniformity and randomness. However, many of these tools are not applicable in many studies due to the following limitations:

Firstly, some of the quality measures are highly subjective to the problem or algorithm at hand. The values of these metrics are sensitive to various factors such as the characteristics of the studied problems, the employed optimization algorithms, and the experiment setup such as the maximum number of objective function evaluations (see Chapter 5 for example).

The final fitness value and success rate are two typical examples of subjective metrics. Their sensitivity limits the generalization capability of the findings to the number of studied problems, employed optimizers, and the levels of variation in parameter settings (*e.g.,* the range and the number of different population sizes that are examined). On the other hand, expanding, studying a large set of candidates is computationally expensive, if even feasible. As a result, the findings from a limited number of experiments are scarcely generalizable to other scenarios.

Secondly, some of the adopted quality metrics are only applicable to specific categories of algorithms. For example, there are well-known measures of randomness, unpredictability, and incompressibility which can be used to evaluate stochastic techniques (see Chapter 3 for more details about stochastic and deterministic methods). However, these tools cannot be used to assess the performance of deterministic techniques due to the lack of a random element in these algorithms.

Thirdly, some measures are computationally expensive [Fang and Lin 2003]. These methods may not be applicable in large-scale domains. Since we are particularly interested in studying the performance of initializers in producing many high-dimensional points, we need to find tools which are efficient regarding memory usage and time complexity.

Considering the above limitations, in this chapter we adopt a subclass of *discrepancy measures* which has analytic formulas. These particular metrics are selected based on three factors:

1. Their results are not affected by the features of benchmarked problems, employed optimizer, and their parameter configuration.

2. We can easily apply them to all kinds of real-value search spaces.

3. They are faster than similar iterative or recursive algorithms which makes them ideal for large-size and high-dimensional populations.

6.2.2 Definitions

The discrepancy value generally indicates the level of nonuniformity of a set of points scattered in a unit hyper-cube [Weyl 1916]. This means more uniform populations have smaller discrepancy values. As mentioned in Chapter 3, uniformity is a desirable property of initial population which plays a crucial role in the effectiveness of population-based metaheuristics when dealing with black-box problems. Therefore, when no prior knowledge is available about the landscape, researchers try to develop new techniques that produce populations having higher uniformity or less discrepancy.

An early version of discrepancy measures is widely known as L_p-discrepancy [Fang and Lin 2003]. The first variants of L_p-discrepancy demand many axillary (randomly generated and uniformly scattered) samples to be able to measure the discrepancy of a given population. This limitation makes them computationally expensive when applied to a large population in a high-dimensional space. Warnock [Fang and Lin 2003], for the first time, proposed a novel analytic formula to compute L_2-discrepancy which resulted in a much faster algorithm. We define the squared L_2-discrepancy of a population $\mathbf{X}$ as follows:

$$D_2(\mathbf{X})^2 = 3^{-D} - \frac{2^{1-D}}{N}\sum_{k=1}^{N}\prod_{l=1}^{D}(1-x_{k,l}^2) + \frac{1}{D^2}\sum_{k=1}^{N}\sum_{j=1}^{N}\prod_{i=1}^{D}[1-\max(x_{k,i},x_{j,i})] \tag{6.1}$$

where D and N are dimensionality and size of the population $\mathbf{X}$, respectively.

To improve the sensitivity and accuracy of L_2-discrepancy, several expansions, such as Star, Modified, Symmetric, Wrap-around, and Centered L_2-discrepancies, have been proposed [Fang and Lin 2003]. For example, the squared Wrap-around discrepancy (WD) measure is defined as:

$$WD(\mathbf{X})^2 = \left(\frac{4}{3}\right)^D + \frac{1}{N^2}\sum_{k=1}^{N}\sum_{j=1}^{N}\prod_{i=1}^{D}\left[\frac{3}{2} - |x_{k,i}-x_{j,i}|(1-|x_{k,i}-x_{j,i}|\right]. \tag{6.2}$$

In this chapter, we mainly focus on the Centered L_2-discrepancy (CD) because it is more accurate in the identification of differences between populations [Hickernell 1998]. The analytic formula of the squared CD is as follows:

$$\begin{aligned} CD(\mathbf{X})^2 = &\left(\frac{13}{12}\right)^D - \frac{2}{N}\sum_{k=1}^{N}\prod_{j=1}^{D}\left(1+\frac{1}{2}|x_{k,j}-0.5| - \frac{1}{2}|x_{k,j}-0.5|^2\right) \\ &+ \frac{1}{N^2}\sum_{k=1}^{N}\sum_{j=1}^{N}\prod_{i=1}^{D}\left[1+\frac{1}{2}|x_{k,i}-0.5| + \frac{1}{2}|x_{j,i}-0.5| - \frac{1}{2}|x_{k,i}-x_{j,i}|\right]. \end{aligned} \tag{6.3}$$

6.3 Experimental Settings

The experimental studies in this chapter comprise of two parts. In the first part, we analyze the trend of population uniformity when a conventional RNG algorithm generates the candidate solutions. We also investigate the effects of population size since it plays an essential role in the homogeneity of the population. In fact, we aim to answer two research questions in this part:

Table 61: Selected Population Initialization Techniques

	Name	Category	Reference
RNG	Mersenne twister	pseudo-random	[Matsumoto and Nishimura 1998]
SIN	sine chaotic map	chaotic number	[Zheng et al. 2013]
TNT	tent chaotic map	chaotic number	[Zheng et al. 2013]
HLT	Halton set	low-discrepancy	[Halton 1960]
SBL	Sobol set	low-discrepancy	[Sobol 1998]
GLP	good lattice point	uniform design	[Sloan and Joe 1994]

1. How much can the uniformity of a population be affected by dimensionality?
2. Is it possible to enhance the uniformity of the initial population in high-dimensional spaces by increasing the population size?

In the second part of the experiments, we compare the performance of alternative initialization techniques with a commonly used RNG technique. We repeat this experiment in a variety of low, medium, and high-dimensional spaces. The questions to be answered in this part are:

1. Can the adoption of alternative initialization techniques improve population uniformity significantly?
2. How sensitive are the alternative initializers to the variation in population size?

To answer these questions, we study three stochastic and three deterministic population initialization techniques in the second part of the experiments. Table 61 lists the chosen techniques. The Chapter 3 provides more information on these techniques.

As mentioned earlier, each time a stochastic technique is executed, it generates a different population due to its dependency to the initial random seed. Therefore, the quality of the resulting set may slightly differ from time to time. To achieve a more solid conclusion, we run each stochastic technique 25 times and average their discrepancies. Note that each run of an algorithm is independent of the other trials because we use exclusive initial seeds each time.

To have similar procedures for both categories, but avoiding the production of the same populations multiple times, we follow *skip* scheme for the deterministic techniques. This scheme generates $25 \times N$ points, but only uses i^{th} N points in the i^{th} run. For example, if $N = 100$, then we generate 2500 points and use the first 100 points in the first series, points 101 to 200 in the second series, and so on. The indexes of the points in the i^{th} trial are easily calculated using the following equations:

$$l_i = (i-1)N + 1, \quad \text{and} \quad u_i = iN. \tag{6.4}$$

where l_i and u_i are the lower and upper bounds of the indexes of points in the i^{th} run, respectively. Note that, in all parts of the experiments, we study 20 different dimension sizes ($2 \leq D \leq 1{,}000$) and 20 unique population sizes ($10 \leq N \leq 10{,}000$).

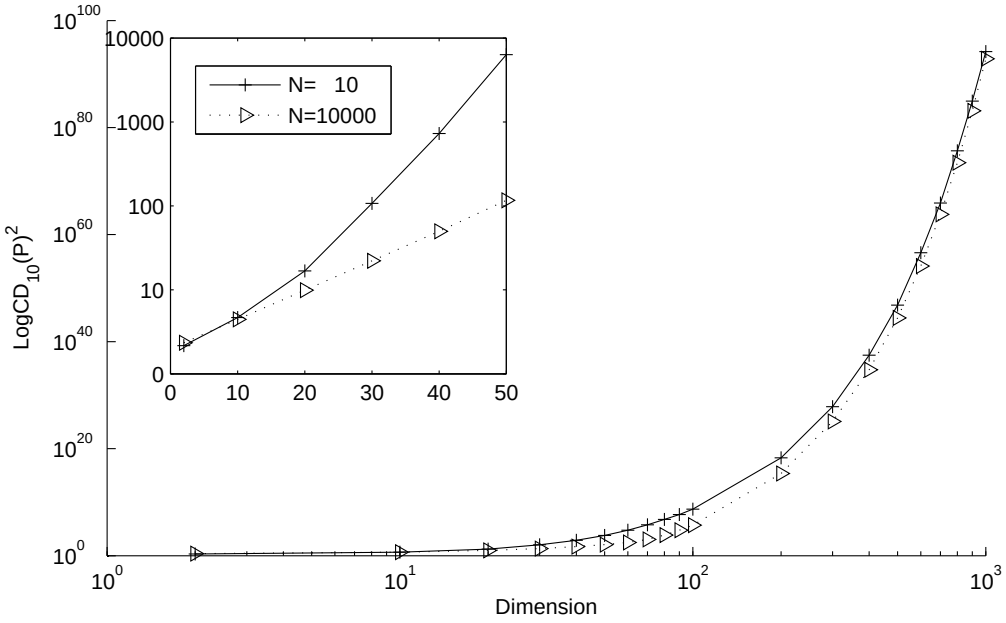


Figure 61: Trend of $CD(\mathbf{X})^2$ of RNG for $2 \leq D \leq 1000$. A part of plot (*i.e.,* $2 \leq D \leq 50$) is zoomed for better demonstration.

6.4 Results and Discussion

6.4.1 Experiment Part (A): Uniformity of RNG points in large-scale

In this part, we compute and compare the CD values of populations generated by the RNG to study the effects of dimensionality and population size on the uniformity.

As Figure 61 shows, CD discrepancy grows (*i.e.,* uniformity drops) exponentially when the dimensionality increases[1]. A closer look at the low dimensional part of the plot (magnified in the top-left corner of the figure) reveals that a large population size may lessen the undesirable effect of high dimensionality. However, the improvement may not be very significant. For example, the CD value of 10,000 points in 50 dimensions is comparable with the CD value of 10 points in 30 dimensions. In other words, 66% growth in dimensionality demands 100,000% increase in population size to recover the uniformity. This issue –widely known as the *curse of dimensionality*– is very critical in large-scale problems.

Figures 62-64 illustrate the effect of population size on the uniformity of small, medium, and large-scale problems, respectively. As Figure 62 indicates, population size has no substantial effect on the uniformity of very small-scale populations, *i.e.,* $D \leq 10$. For higher dimensions, especially for $30 \leq D \leq 50$, population size has a significant effect on uniformity such that it can be improved 10 to 20 times in the CD scale. However, the magnitude of improvements falls rapidly such that increasing population size beyond 1,000 points shows only a minimal improvement. In other words, it is reasonable to increase population size for the problems of size 20 to 50, but not beyond 1,000 points.

Figure 63 demonstrates similar pattern for medium-scale spaces. The only difference is the slopes of the curves which are slower for these spaces. Therefore, a considerable improvement is expected even for population size beyond 3,000 points. In other words,

[1] $\mathcal{P}$ in the plot refers to $\mathbf{X}$ in the equations.

having larger populations in medium-scale problems is reasonable when computationally feasible. As the plotted results indicate, the improvement in uniformity in these spaces also drops rapidly.

Homogeneity metrics in a high-dimensional space are much worse than small and medium-scale problems. As Figure 64 reveals, the uniformity of populations in spaces of above 100 dimensions is such low that increasing population size from 1,000 to 10,000 cannot improve it in a meaningful scale. Having a closer look at the plot shows that the reasonable population size for such large-scale problems is surprisingly less than 1,000 points. Note that this does not imply the population size does not affect large-scale problems; instead, it means the population size must be astronomically large to achieve a significant enhancement in the uniformity. Since evaluating high-dimensional populations in that magnitude is currently computationally expensive, keeping it around 1,000 points is more practical and reasonable.

6.4.2 Experiment Part (B): Comparison studies

To compare alternative initialization techniques with a common RNG, we propose a simple formula reflecting relative improvement achieved from each of these techniques:

$$\textbf{\% improvement} = \frac{log_{10}\, CD(\mathbf{X}_c)^2 - log_{10}\, CD(\mathbf{X}_i)^2}{log_{10}\, CD(\mathbf{X}_c)^2} \times 100 \tag{6.5}$$

where $\mathbf{X}_c$ is the population generated by the control technique, RNG, and $\mathbf{X}_i$ is the population produced by the i^{th} alternative initialization technique. In cases that an alternative method functions worse than the baseline, the value of Equation (6.5) will be less than zero.

As Figure 65 reveals, some techniques such as TNT and SBL are successful in improving the baseline, although the most significant improvement in $2 \leq D \leq 50$ is less than 20%. Another observation from this plot is that while some techniques *e.g.,* GLP are very sensitive to population size (compare the dash-square and solid-square trends),

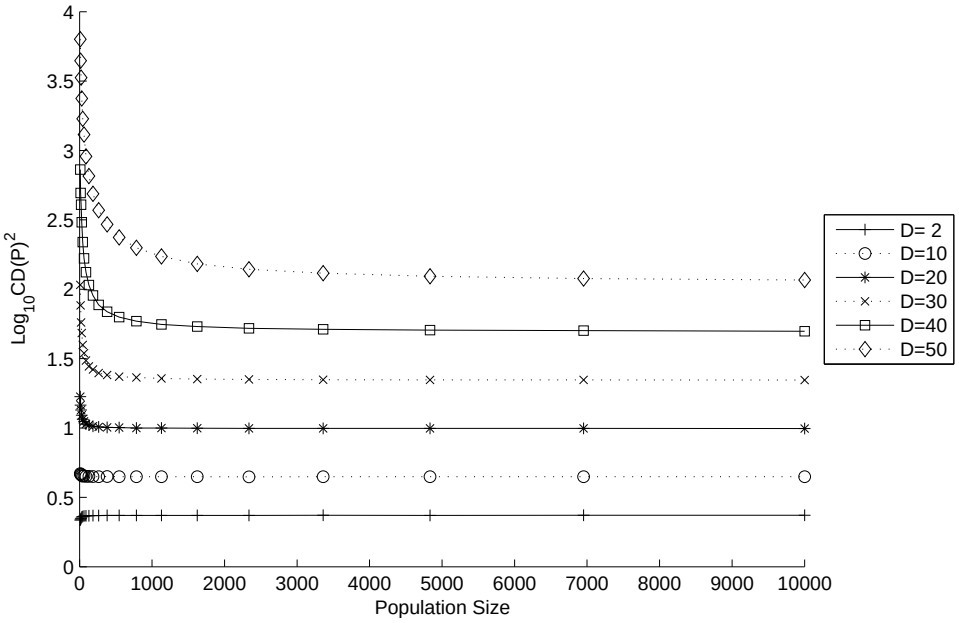


Figure 62: Effect of population size on $CD(\mathbf{X})^2$ of RNG in low dimensions ($D \leq 50$)

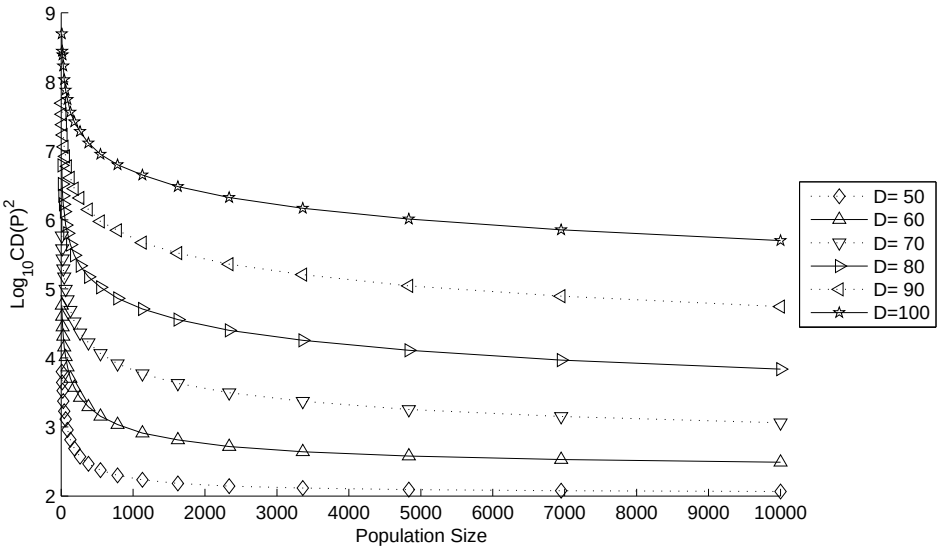


Figure 63: Effect of population size on $CD(\mathbf{X})^2$ of RNG in medium dimensions

others including SBL are more stable. However, with no exception, all techniques perform relatively better when the population size increases. Figure 65 also shows that mixed good and bad results can be expected from both categories of initialization techniques which confirms the findings from Chapter 5. Note that the negative numbers indicate detrimental effects.

Figure 66 depicts the improvements gained from advanced techniques in the medium and high-dimensional spaces. As can be seen in the plot, all trends converge to one of the three values: 0%, −25% and, −80%. This clearly shows that employing advanced initialization techniques provides no significant improvement in high-dimensions, at least regarding uniformity. Even increasing population size from 10 to 10,000 does not result in a significant improvement. Figure 66 also shows SBL with 10 and TNT with both 10 and

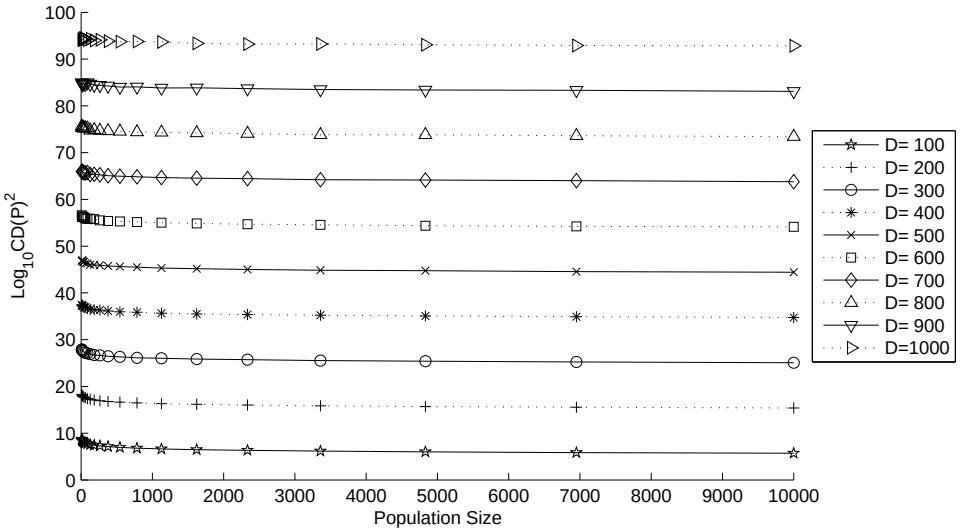


Figure 64: Effect of population size on $CD(\mathbf{X})^2$ of RNG in high dimensions

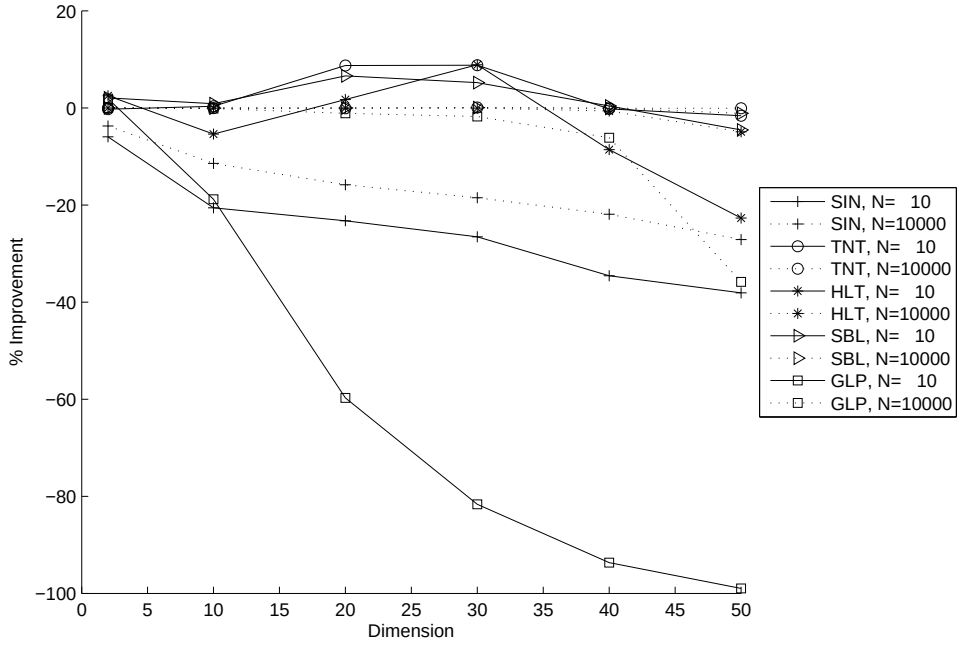


Figure 65: Improvements gained from advanced techniques in low dimensions

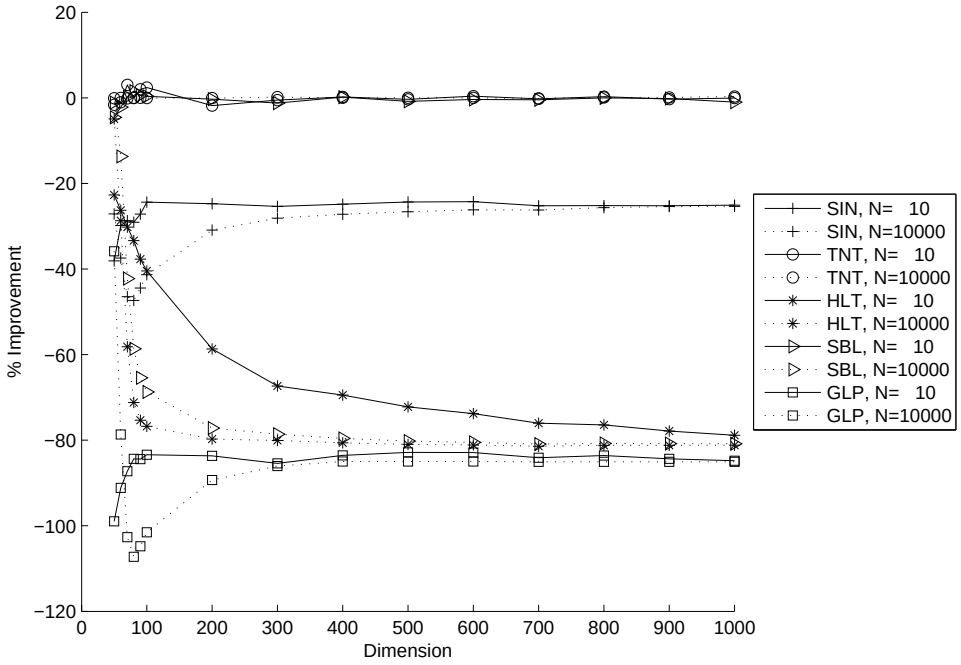


Figure 66: Improvements gained from advanced techniques in medium and high dimensions

10,000 population sizes perform virtually the same as RNG. The others, however, perform poorly in comparison with an RNG with the same population size.

In some cases, including SBL, an increase in the population size decreases the relative improvement. We do not interpret this as an adverse effect of population size on these algorithms. Note that the plot illustrates the relative improvements which are calculated based on the performance of RNG (see Equation (6.5)). Therefore, a decrease in relative improvement indicates that expanding the population size has a more significant positive impact on RNG than these algorithms such that the relative improvement metric drops when population size grows.

6.5 Chapter Summary

In this chapter, we investigated the reasons that cause alternative population initialization techniques to not performing as expected in high-dimensional spaces. We used Centered L_2-discrepancy as a generic uniformity measure to study the effects of dimensionality (from $D = 2$ to $D = 1{,}000$) on conventional and alternative initializers where population size varies from 10 to 10,000 points.

Our investigations show that the uniformity of initial population drops exponentially when dimensionality rises linearly. Low uniformity, which signals a weak coverage and low diversity, degrades the quality of initial populations dramatically. Except for some small and medium-scale spaces, even increasing population size up to a computationally feasible limit cannot maintain the uniformity of the initial population.

Our experimental results reveal that the alternative initializers are as vulnerable to the curse of dimensionality as simple random number generators. Therefore, adopting such techniques in medium and large-scale spaces may not necessarily result in a statistically significant improvement. In this regard, some alternative techniques are even more susceptible to the adverse effect of dimensionality than the simple pseudo-random number generators. Accordingly, we only recommend the use of alternative techniques when the population and dimension sizes are manageable. In higher dimensional spaces or when the population size is relatively large, no significant improvement is excepted from the advanced techniques.

Part II

Imbalanced Problems

CHAPTER 7

Resource Allocation in Cooperative Coevolution

It can be very practical to divide a complex optimization problem into simpler subproblems and solve them using conventional optimizers. Very often, these subproblems exhibit different features which means solving each of them will have a different level of contribution in solving the main large-scale problem. This challenges the uniform resource distribution devised in traditional Cooperative Coevolutionary (CC) algorithms. We devote this chapter to explore the issue of unequal contributions from the resource allocation perspective. We also look into some of the available solutions and summarize their strengths and weaknesses. In the next chapters, we will develop this line of research further.

7.1 Introduction

Many large-scale optimization tasks can be modeled as modular problems which consist of two or more isolated or loosely coupled subfunctions [Bouaricha and Morè 1997, Colson and Toint 2005]. Dividing a high-dimensional problem into a set of ideally disconnected components helps practitioners to apply a wide range of Cooperative Coevolutionary (CC) algorithms and unleash the power of parallel processing to solve them more effectively and efficiently [Lescrenier 1988].

Most of the decomposition-based metaheuristics, including the CC models, assume that all subfunctions are equally important and solving each of them is equally challenging. As a result, they allocate the computational budget (in terms of population size and the number of iterations) uniformly across all subfunctions. In practice, however, it is very likely that the components of a problem to have unequal levels of importance with respect to their contributions to the overall fitness improvement [Omidvar et al. 2011]. In other words, solving some of the subproblems will result in a considerably larger impact on the progress of the optimization than the other components.

The *'impact of a component on the overall fitness improvement'* has been defined as a measure of the component's contribution. In the literature, the problems that consist of components with unequal contributions are referred as *imbalanced functions*. The root cause of such imbalance can be any combination of nonuniform dimensionality, nonidentical search landscape, and unequal coefficients (especially in partially additive separable problems).

It is not difficult to show that the uniform resource allocation that is common in traditional decomposition-based techniques may not perform well in dealing with imbalanced functions. Indeed, in cases where the computational budget is limited, it is more rewarding to make an extra effort on solving subproblems with the largest contribution [Yang et al. 2017]. Otherwise, spending the same amount of resources irrespective of the magnitudes of contributions made by the components will result in a waste of valuable computational budget. This means we may experience an early termination before being able to solve the problem to a desirable extent.

Several attempts have been made to address the imbalance contributions especially when a CC technique is adopted [Omidvar et al. 2011, Mahdavi et al. 2016c, Omidvar et al. 2016, Yang et al. 2017]. These techniques, which we call Contribution-Aware CCs (CACCs) hereafter, aim to find the most contributing components of a black-box problem and adjust the budget allocation according to their contribution. We review the most widely used CACCs in the following sections.

7.2 Formal Definition

In the field of optimization, the issue of unequal contribution of subfunctions is called *imbalance problem* (*a.k.a.* imbalance issue) [Omidvar et al. 2011, Li et al. 2013a]. Therefore, a function that exhibits such imbalance feature is called an *imbalanced function* [Omidvar et al. 2015; 2016]. Definition 14 provides a formal definition of imbalanced functions.

Definition 14 (Imbalanced Function)**.** Let f is a partially additively separable function:

$$f(\mathbf{x}) = \sum_{k=1}^{K} C_k \cdot f_k(\mathbf{x}_k), \tag{7.1}$$

where $K > 1$ is the number of components, f_k denotes the k^{th} subfunction ($|\mathbf{x}| = \sum_{k=1}^{K} |\mathbf{x}_k|$), and $C_k \in \mathbb{R}$ are arbitrary constant coefficients. Then, f is an *imbalanced function* if $\exists i \in \{1, \ldots, K\}$ and $\exists j \in \{1, \ldots, K\}$ that $i \neq j$ and at least one of the following is true:

$$\begin{array}{ll} C_i \neq C_j, & \text{or} \\ |f_i| \neq |f_j|, & \text{or} \\ f_i \neq f_j. & \end{array}$$

In the above definition, $f_i \neq f_j$ indicates a case in which the landscapes of two components exhibit different features. Also note that, $C_i \neq C_j$ is only valid as a criteria when $f_i = f_j$. Otherwise, the imbalance in the coefficients can be canceled out by the differences in the function definition (for example if $f_i = \frac{C_j}{C_i} f_j$, then $C_i \cdot f_i = C_j \cdot f_j$ although $C_i \neq C_j$).

7.3 Cooperative Coevolutionary Algorithms

As mentioned in Section 2.5, decomposing a complex problem into smaller subproblems is an effective approach to reduce the adverse influence of dimensionality on the performance of optimization techniques. One of the subclasses of metaheuristics that benefits from this *divide-and-conquer* approach is the category of Cooperative Coevolutionary (CC) algorithms.

A typical round-robin CC algorithm usually works as follows. Consider a large-scale function f is decomposed (either manually or using a variable grouping algorithm) into K

Algorithm 1 Cooperative Coevolution Framework

1: **function** $\texttt{CC}(f, D, N)$
2: $(\mathbf{X}, \mathbf{f}) \leftarrow \texttt{initialization}(f, N, D)$ ▷ Initialization
3: $(\mathcal{D}, K) \leftarrow \texttt{initGrouping}(f)$ ▷ Decomposition
4: $\vec{\mathbf{x}} \leftarrow \texttt{initContextVector}(\mathbf{X}, \mathbf{f})$ ▷ Initial *Context*
5: $k \leftarrow 0$ ▷ Initial round-robin
6: **while** $\texttt{termination}() \neq \texttt{True}$ **do** ▷ Main loop
7: $k \leftarrow (k \mod K) + 1$ ▷ Select next component
8: $(\mathbf{X}, \mathbf{f}) \leftarrow \texttt{optimize}(f, \mathbf{X}, \vec{\mathbf{x}}, \mathcal{D}, k, \mathbf{f}, \delta_t)$ ▷ One epoch optimization
9: $\vec{\mathbf{x}} \leftarrow \texttt{updateContextVector}(\mathbf{X}, \mathbf{f}, \vec{\mathbf{x}})$ ▷ Update Context Vector
10: $\mathcal{D} \leftarrow \texttt{updateGrouping}(f, \mathcal{D})$ ▷ Dynamic decomposition
11: **return** $\mathbf{X}[\arg\min(\mathbf{f}), :]$ ▷ Return final solution

12: **function** $\texttt{optimize}(f, \mathbf{X}, \vec{\mathbf{x}}, \mathcal{D}, k, \mathbf{f}, g)$
13: **for** $r \leftarrow 1 \cdots N$ **do** ▷ Replicate $\vec{\mathbf{x}}$ N times
14: $\vec{\mathbf{X}}[r, :] \leftarrow \vec{\mathbf{x}}$
15: $\vec{\mathbf{X}}[:, \mathcal{D}[k]] \leftarrow \mathbf{X}[:, \mathcal{D}[k]]$ ▷ Plug k^{th} component
16: $(\vec{\mathbf{X}}, \mathbf{f}) \leftarrow \texttt{optimizer}(f, \vec{\mathbf{X}}, g)$ ▷ 1 epoch optimization
17: $\mathbf{X}[:, \mathcal{D}[k]] \leftarrow \vec{\mathbf{X}}[:, \mathcal{D}[k]]$ ▷ Update solutions
18: **return** $(\mathbf{X}, \mathbf{f})$ ▷ Return new subpopulation

f_k as in shown in Equation (7.1). Then, each component f_k is treated as a separate optimization problem and tackled by an arbitrary optimizer (*a.k.a.* component/subproblem optimizer). Since the exact formula or simulator of every single subproblem might not be available, the fitness of each candidate subsolution is evaluated using some information from other components. For example, a so-called *context vector* may be constructed by merging the potential subsolutions of all subproblems. In other words, since we cannot evaluate $f_k(\mathbf{x}_k)$, the CC algorithm needs to combine K different subsolutions $\mathbf{x}_k$, each of which corresponds to a different component, to construct one D-dimensional context vector $\vec{\mathbf{x}}$ and evaluate $f(\vec{\mathbf{x}})$.

The variations in the fitness values of the context vector over time can provide an accurate estimate of the the changes in the fitness values of the recently updated subsolutions. For example, let f be an additively separable function and $\Delta_f^{(t)} = f(\vec{\mathbf{x}}^{(t)}) - f(\vec{\mathbf{x}}^{(t-1)})$ be the difference between two consecutive evaluations of f when only the k^{th} component of $\vec{\mathbf{x}}$ has been changed since time t. Then, $\Delta_f^{(t)} = f_k(\mathbf{x}_k^{(t)}) - f_k(\mathbf{x}_k^{(t-1)})$. Therefore, CCs must only change one of the components at a time and keep the rest of $\vec{\mathbf{x}}$ fixed. Otherwise, multiple components may contribute in the value of $\Delta_f^{(t)}$.

Traditionally, each subproblem is optimized for one *epoch* at a time which consists of one or more optimization iteration. After each epoch, the context vector is updated. When all subproblems are optimized for one epoch, one *coevolutionary cycle* is completed. This cycle is repeated until the termination criteria are met. At the end, the best subsolution for each subproblem is selected and merged with the other subsolutions to form a D-dimensional solution to the main problem.

Algorithm 1 shows the main steps of the round-robin CC framework. Here, N and T are the population size and the maximum number of epochs, respectively. Therefore, $\mathbf{X}$ is $N \times D$ matrix. For the sake of simplicity, we omit some of the control parameters (*e.g.,* optimizer's parameters in line 16).

The population initialization, problem decomposition, and context vector creation all

are performed in lines 2–4. The initial population and their fitness are stored in $\mathbf{X}$ and $\mathbf{f}$. The flexibility of CC framework allows the practitioners to initialize the context vector in many different ways. However, we usually select the best solution in the initial population as the initial context vector.

In Algorithm 1, as well as other algorithms in this dissertation, the numbers in the brackets represent the row and column indices of a matrix. Therefore, $\mathbf{A}[i, j]$ indicates the element that is located at the i^{th} row and the j^{th} column. A colon mark substituting a row or column index indicates the whole row or column. For example, $\mathbf{A}[:, j]$ means all rows of the j^{th} column of matrix $\mathbf{A}$.

At line 3 of the algorithm, the D-dimensional function f is divided into K subfunctions. Note that $\mathcal{D}$ is a set of sets. Indeed, $\mathcal{D}$ consists of K smaller sets $\mathcal{D}_k$, each of which stores d_k indices of the decision variables that form the k^{th} subfunction. Therefore, $\bigcup_{k=1}^{K} \mathcal{D}_k = \{1, \ldots, D\}$ and hence $\sum_{k=1}^{K} |\mathcal{D}_k| = D$.

The main loop starts at line 6. At line 7, the next subfunction is selected in a round-robin fashion and optimized in line 8 for one epoch which consists of δ_t optimization iterations. As shown in the definition of `optimize`, only the columns that correspond to the k^{th} components are modified here whilst other parts of $\mathbf{X}$ remain intact. Then, the population is re-evaluated and the context vector is updated typically by plugging the best subsolution fo the k^{th} subfunction into the old context vector (*i.e.,* $\vec{\mathbf{x}}[\mathcal{D}_k] \leftarrow \mathbf{X}[\arg\min(\mathbf{f}), :]$ in a minimization problem). In the case of dynamic decomposition, the variable grouping should be updated at line 10. Then, the main loop continues for another cycle if the termination criteria have not been satisfied yet. Otherwise, the final solution, which is the combination of all the best subsolutions, is returned.

From the budget management point of view, the bottleneck of the CC framework is the way it selects the next subfunction to solve. As shown in line 7 of Algorithm 1, round-robin CCs simply select the next components solely based on their indices. In other words, they uniformly distribute the remaining resources regardless of the past contribution or projected future performance of the subproblems. The CACCs that we review in the next section try to enhance this part of traditional CC framework.

7.4 Contribution-Aware Cooperative Coevolution

In this section, we review the researches that try expand round-robin CC framework by proposing Contribution-Aware CC (CACC) algorithms. This provides us a foundation to propose our solutions in the following chapters.

7.4.1 Contribution-Based Cooperative Coevolution 1 and 2

The so-called Contribution-Based CC (CBCC) technique is the first published work that explicitly addresses the imbalance contributions of subfunctions in the context of large-scale optimization [Omidvar et al. 2011]. This framework has two variants: CBCC1 and CBCC2. These algorithms adopt simple heuristics to find the most contributing component and optimize it more often than the others.

The CBCC framework consists of two phases, namely (*exploration* and *exploitation*), which are executed iteratively. The exploration phase is similar to one cycle of a round-robin CC where all components are optimized for a single epoch. The only difference is that CBCCs record the fitness values of the best solutions before and after optimization epoch. The distance between these values is used as an indicator of a component's contribution

Algorithm 2 Contribution-Based Cooperative Coevolution 1 and 2

function `CBCC`(f, D, N, v)
  $(\mathbf{X}, \mathbf{f}_c) \leftarrow$ `initialization`(f, N, D) ▷ Initialization
  $(\mathcal{D}, K) \leftarrow$ `initGrouping`(f) ▷ Decomposition
  $\vec{\mathbf{x}} \leftarrow$ `initContextVector`$(\mathbf{X}, \mathbf{f})$ ▷ Initial *Context*
  $\boldsymbol{\mu} \leftarrow$ `zeros`$(1, K)$ ▷ Initial contributions
  while `termination`$() \neq$ `True` **do** ▷ **Main loop**
    for $k \leftarrow 1 \cdots K$ **do** ▷ **Exploration Phase**
      $\mathbf{f}_p \leftarrow \mathbf{f}_c$
      $(\mathbf{X}, \mathbf{f}_c, \vec{\mathbf{x}}, \boldsymbol{\mu}) \leftarrow$ `optimize`$(f, \mathbf{X}, \vec{\mathbf{x}}, \mathcal{D}, k, \mathbf{f}_p)$ ▷ 1 epoch optimization
    $\overset{*}{k} \leftarrow \arg\max \boldsymbol{\mu}$ ▷ Select a component
    while $\min \mathbf{f}_c < \min \mathbf{f}_p$ **do** ▷ **Exploitation Phase**
      $\mathbf{f}_p \leftarrow \mathbf{f}_c$
      $(\mathbf{X}, \mathbf{f}_c, \vec{\mathbf{x}}, \boldsymbol{\mu}) \leftarrow$ `optimize`$(f, \mathbf{X}, \vec{\mathbf{x}}, \mathcal{D}, k, \mathbf{f}_p)$ ▷ 1 epoch optimization
      if $v =$ `CBCC1` **then** ▷ `CBCC1` or `CBCC1`?
        break
  return $\mathbf{X}[\arg\min(\mathbf{f}_c), :]$ ▷ Return final solution

function `optimize`$(f, \mathbf{X}, \vec{\mathbf{x}}, \mathcal{D}, k, \mathbf{f}_p)$
  for $r \leftarrow 1 \cdots N$ **do** ▷ Replicate $\vec{\mathbf{x}}$ N times
    $\vec{\mathbf{X}}[r, :] \leftarrow \vec{\mathbf{x}}$
  $\vec{\mathbf{X}}[:, \mathcal{D}[k]] \leftarrow \mathbf{X}[:, \mathcal{D}[k]]$ ▷ Plug k^{th} component
  $(\vec{\mathbf{X}}, \mathbf{f}_c) \leftarrow$ `optimizer`$(f, \vec{\mathbf{X}}, \delta_t)$ ▷ 1 epoch optimization
  $\mathbf{X}[:, \mathcal{D}[k]] \leftarrow \vec{\mathbf{X}}[:, \mathcal{D}[k]]$ ▷ Update solutions
  $\vec{\mathbf{x}} \leftarrow$ `updateContextVector`$(\mathbf{X}, \mathbf{f}_c, \vec{\mathbf{x}})$ ▷ Update *Context*
  $\boldsymbol{\mu}[k] \leftarrow \boldsymbol{\mu}[k] + \min \mathbf{f}_p - \min \mathbf{f}_c$ ▷ Update contribution
  return $(\mathbf{X}, \mathbf{f}_c, \vec{\mathbf{x}}, \boldsymbol{\mu})$ ▷ Return to `CBCC`

to improving the overall objective value. In the next exploitation phase, the component with the highest accumulated contribution receives more computational resources.

The exploitation phase of CBCC1 consists of one optimization epoch of the most contributing component. In contrast, CBCC2 continuously optimizes this component until there is no more improvement in the fitness of the best solution found so far. These algorithms switch between exploration and exploitation phases until the termination criteria are met.

In Chapter 8, we conduct a series of sensitivity analysis on CBCCs to better understand them. Then, based on the findings, we develop a generation of CBCC framework called CBCC3 in the same chapter. Finally, in Chapter 9 we propose a new set of benchmarking suite to deeply study the budget allocation performance of CBCC family. This provides us the foundation to develop a general framework that brings resource allocation into CC framework. We call this framework Bandit-Based CC (BBCC) and show that all previously proposed CACCs can be formulated as special instances of the general BBCC framework.

Algorithm 2 explains the CBCC steps in details. The first few lines (*i.e.,* the initialization steps) prior to the start of the main loop is very similar to the round-robin CC pseudocode that we presented in Algorithm 1. The only difference is the introduction of $\boldsymbol{\mu}$ which is a vector of size K that stores the accumulated improvement received by each component. These values are then treated as estimations of components contributions

and used in selecting the most influencing subfunction.

The first part of the main loop represents the exploration phase where all subfunctions are optimized for exactly one epoch. The `optimize` that is called at lines 9 and 13 of Algorithm 2 is very similar to `optimize` that is defined in Algorithm 1 (see lines 12–18) except we also update the value of μ_k after the optimization of the k^{th} subfunction by an arbitrary optimizer at line 21. As line 24 shows, the amount of recent improvement is calculated based on the difference between the objective values of the current best solution and the previous best solution. More formally:

$$\mu_k^{(t)} = \mu_k^{(t-1)} + \overset{*}{f}_p - \overset{*}{f}_c \tag{7.2}$$

where $\overset{*}{f}_c = \min(\mathbf{f}_c)$ and $\overset{*}{f}_p = \min(\mathbf{f}_p)$.

Both CBCC variants select the component with the highest estimated contribution at line 10. Then, they give it at least one more epoch of optimization. In the case of CBCC1, the loop breaks after exactly one extra epoch while CBCC2 continues optimizing the selected component ($\overset{*}{k}$) until it observes no improvement (*i.e.*, $\overset{*}{f}_c \not< \overset{*}{f}_p$)).

7.4.2 Cooperative Coevolution with Adaptive Optimizer Iterations

As mentioned earlier, CBCC1 and CBCC2 estimate the contribution of subfunctions based the historical observations. Accumulating the improvement from the first trial to the end makes CBCCs being very slow in adapting to the changes. For example, both CBCCs variants need a long time to trigger a switch to another component when the most contributing component stops improving (either because it already solved or just stagnated). The CC with Adaptive Optimizer Iterations (CCAOI) is proposed to remedy this problem [Trunfio 2015].

In brief, CCAOI calculates a real-value number γ using a formula inspired by the Gini's index of inequality. The value of γ indicates the overall severity of the imbalance in the contributions of subfunctions. The value of γ is always bounded between 0 and 1 where $\gamma = 0$ suggests a perfect balance function whereas $\gamma = 1$ indicates the most extreme imbalance scenario.

Unlike CBCCs, the CCAOI does not offer fixed-length epochs. Instead, it guarantees that in an optimization cycle a component will be optimized for at least G_m and at most G_M evolutionary generations. More precisely, the k^{th} component will be activated for g_k optimization iterations which is calculated as follows:

$$g_k = G_m + \frac{\gamma G_M \dot{\delta}_k}{K\bar{\delta}} \tag{7.3}$$

where $\dot{\delta}_k$ is the latest normalized contribution of the k^{th} subfunction and $\bar{\delta}$ is the average of the recent contributions:

$$\bar{\delta} = \frac{1}{K}\sum_{k=1}^{K} \dot{\delta}_k \tag{7.4}$$

The Gini's index of inequality (γ in Equation (7.3)) is computed according to the following formula:

$$\gamma = \frac{\sum_{k=1}^{K}\sum_{\acute{k}=1}^{K} |\dot{\delta}_k - \dot{\delta}_{\acute{k}}|}{2K^2\bar{\delta}} \tag{7.5}$$

The value of $\dot{\delta}_k$ in the above equations is calculated as:

$$\dot{\delta}_k = \begin{cases} \frac{\overset{*}{f}_p - \overset{*}{f}_c}{g_k} & \text{if } \overset{*}{f}_c < \overset{*}{f}_p \\ 0 & \text{otherwise,} \end{cases} \tag{7.6}$$

Algorithm 3 Cooperative Coevolution with Adaptive Optimizer Iterations

1: **function** CCAOI(f, D, N, G_m, G_M)
2: $(\mathbf{X}, \mathbf{f}_c) \leftarrow \texttt{initialization}(f, N, D)$ ▷ Initialization
3: $(\mathcal{D}, K) \leftarrow \texttt{initGrouping}(f)$ ▷ Decomposition
4: $\vec{\mathbf{x}} \leftarrow \texttt{initContextVector}(\mathbf{X}, \mathbf{f}_\mathbf{p})$ ▷ Initial *Context*
5: $\dot{\boldsymbol{\delta}} \leftarrow \texttt{zeros}(1, K)$ ▷ Initial contributions
6: **while** $\texttt{termination}() \neq \texttt{True}$ **do** ▷ **Main loop**
7: $\bar{\delta} \leftarrow (\sum_{k=1}^{K} \dot{\boldsymbol{\delta}}[k])/K$ ▷ Equation (7.4)
8: $\gamma \leftarrow (\sum_{k=1}^{K} \sum_{\acute{k}=1}^{K} |\dot{\boldsymbol{\delta}}[k] - \dot{\boldsymbol{\delta}}[\acute{k}]|)/(2K^2\bar{\delta})$ ▷ Equation (7.5)
9: **if** $\bar{\delta} > 0$ **then** ▷ **Resource allocation**
10: **for** $k \leftarrow 1 \cdots K$ **do**
11: $\mathbf{g}[k] \leftarrow G_m + (\gamma G_M \dot{\boldsymbol{\delta}}[k])/(K\bar{\delta})$ ▷ Equation (7.3)
12: **for** $k \leftarrow 1 \cdots K$ **do** ▷ **Optimization cycle**
13: $\mathbf{f}_p \leftarrow \mathbf{f}_c$
14: $(\mathbf{X}, \mathbf{f}_c) \leftarrow \texttt{optimize}(f, \mathbf{X}, \vec{\mathbf{x}}, \mathcal{D}, k, \mathbf{f}_p, \mathbf{g}[k])$ ▷ 1 epoch optimization
15: $\vec{\mathbf{x}} \leftarrow \texttt{updateContextVector}(\mathbf{X}, \mathbf{f}_\mathbf{c}, \tilde{\mathbf{x}})$ ▷ Update Context Vector
16: $\dot{\boldsymbol{\delta}}[k] \leftarrow \max(0, \min(\mathbf{f}_p) - \min(\mathbf{f}_c))/\mathbf{g}[k]$ ▷ Equation (7.6)
17: **return** $\mathbf{X}[\arg\min(\mathbf{f}_c), :]$ ▷ Return final solution

where $\overset{*}{f}_c = \min(\mathbf{f}_c)$ and $\overset{*}{f}_p = \min(\mathbf{f}_p)$.

Algorithm 3 shows the main steps of CCAOI. The first few lines of CCAOI pseudocode are very similar to CCBC steps that are presented in Algorithm 2. In the beginning of main loop that starts at line 6 we need to calculate the average normalized contribution $\bar{\delta}$ and Gini's index γ based on Equation (7.4) and Equation (7.5), respectively. Then, at line 11, CCAOI updates the number of iterations each subfunction should be optimized based on Equation (7.3) only if the current average of contributions is positive. Whether g_k is being updated or not, the CCAOI's optimization cycle always starts at line 12.

There are two key differences between CCAOI's optimization cycle and CBCCs' exploitation phase. Firstly, CBCCs only try to optimize the most contributing component they have found (*i.e.,* $\overset{*}{k}$) and do not touch the other subfunctions in this phase. In contrast, CCAOI optimizes all subfunctions for at least G_m iterations. Secondly, the number of generations passed to improve a component is fixed in CBCC framework (similar to general CC framework). It means each selected component will be optimized for exactly one epoch where the epoch length $\delta_t$is always fixed. Conversely, CCAOI does not appreciate the fixed-length epoch concept; it may assign any arbitrary number of generations to a subfunction as long as this value is in the predefined range (*i.e.,* $G_m \leq g_k \leq G_M$).

In theory, the aforementioned features of CCAOI make it more flexible than its CBCC counterparts. In practice, however, it has been only applied to a limited number of problems from CEC'10 and not compared against other CACCs. Therefore, it remains unclear to what extent CCAOI improves CBCCs.

7.4.3 Multilevel Optimization Framework Based on Variables Effect

The Multilevel Optimization Framework Based on Variables Effect (MOFBVE) is another attempt to address the imbalance problem [Mahdavi et al. 2016c]. Particularly, MOFBVE tries to identify the mutual effect of decision variables of black-box problems and then employ this knowledge to improve the CC framework. To achieve its goal, MOFBVE adopts a special sensitivity analysis tool called Morris Screening to measure the variables

Algorithm 4 Multilevel Optimization Framework Based on the Variable Effect

1: **function** MOFBVE(f, D, N, K)
2: $(\mathbf{X}, \mathbf{f}) \leftarrow$ initialization(f, N, D) ▷ Initialization
3: $(\overset{*}{\boldsymbol{\mu}}, \overset{*}{\boldsymbol{\sigma}}) \leftarrow$ Morris(f, D) ▷ Sensitivity analysis
4: $\mathcal{D} \leftarrow$ KMeans$(\langle \overset{*}{\boldsymbol{\mu}}, \overset{*}{\boldsymbol{\sigma}} \rangle, K)$ ▷ Decomposition
5: $\vec{\mathbf{x}} \leftarrow$ initContextVector$(\mathbf{X}, \mathbf{f})$ ▷ Initial *Context*
6: **for** $k \leftarrow 1 \cdots K$ **do**
7: $\mathbf{d}[k] \leftarrow$ size$(\mathcal{D}[k])$ ▷ Cluster sizes
8: $\mathbf{g}[k] \leftarrow 10\mathbf{d}[k]$ ▷ Allocate resources
9: $\bar{\boldsymbol{\mu}}[k] \leftarrow \sum_{i=1}^{\mathbf{d}[k]} \overset{*}{\boldsymbol{\mu}}[i] / \sum_{i=1}^{D} \overset{*}{\boldsymbol{\mu}}[i]$ ▷ Compute effect
10: $\grave{\mathbf{k}} \leftarrow$ sort$(\mathcal{D}, \bar{\boldsymbol{\mu}})$ ▷ Sort components
11: **for** $\grave{k} \in \grave{\mathbf{k}}[1 : K-1]$ **do**
12: $(\mathbf{X}, \mathbf{f}, \vec{\mathbf{x}}) \leftarrow$ optimize$(f, \mathbf{X}, \vec{\mathbf{x}}, \mathcal{D}, \grave{k}, \mathbf{f}, \mathbf{g}[\grave{k}])$ ▷ Optimize $\grave{k}^{\text{th}}$ group
13: $(\mathbf{X}, \mathbf{f}) \leftarrow$ optimizer$(f, \mathbf{X}, \mathbf{f}, \mathbf{g}[\grave{k}])$ ▷ Optimize all variables
14: $\vec{\mathbf{x}} \leftarrow$ updateContextVector$(\mathbf{X}, \mathbf{f})$ ▷ Update *Context*
15: **while** termination() $\neq$ True **do**
16: $(\mathbf{X}, \mathbf{f}) \leftarrow$ optimizer$(f, \mathbf{X}, \mathbf{f}, 1)$ ▷ Optimize all variables
17: **return** $\mathbf{X}[\arg\min(\mathbf{f}), :]$ ▷ Return final solution

effect and group them accordingly [Morris 1991]. Similar sensitivity analysis tools have been used in the literature to identify the input parameters of a model with the most significant impact on its output(s) [Saltelli et al. 2008].

In brief, MOFBVE works as follows. It employs Morris Screening algorithm on the main function to calculate two sensitivity metrics $\overset{*}{\mu}$ and $\overset{*}{\sigma}$ at the early stage of the optimization. Then, the variables with similar effects on the fitness function are grouped using a conventional K-Means clustering algorithm [MacQueen et al. 1967]. Next, it calculates the contribution of each group of variables using:

$$\bar{\mu}_k = \frac{\sum_{i=1}^{D_k} \overset{*}{\mu}_i}{\sum_{i=1}^{D} \overset{*}{\mu}_i} \tag{7.7}$$

where D_k is the number of variables in the k^{th} group (*i.e.*, dimension size of the k^{th} component), D is the size of the main objective function, and $\bar{\mu}_k$ denotes the estimated contribution of the k^{th} component.

MOFBVE ranks the subfunctions based on their $\bar{\mu}$ value such that the group with the highest value in $\bar{\mu}$ will be optimized before any other subfunction. The portion of resources each subproblem receives is proportional to its dimensionality.

Algorithm 4 provides the MOFBVE steps in more details. As the pseudocode shows, MOFBVE starts with initializations followed by Morris Screening and K-Means Clustering (see lines 1– 5). Then, it computes the size, budget, and ranks of each component (see lines 6–9). Next, MOFBVE sorts the components based on their ranks in a descending order (see line 10). The symbol $\grave{k}$ in Algorithm 4 denotes the index of components after sorting. These indices are then used at lines 12–13 to optimize the components from the most to the least contributing. Note that each component is optimized for $g_{\grave{k}}$ which is equivalent to 10 times its dimension size (see line 8). After component optimization at line 12, all D variables of the main problem are optimized for the same number of iterations (line 13), and then, the context vector is updated (line 14). Finally, all the remaining computational budget is spent to optimize the D-dimensional function f (line 16).

There are a few bold factors that make MOFBVE different from other CACCs. Firstly, it does not employ any conventional decomposition technique to divide the problem into subproblems. Instead, it groups the variables into K groups using K-Means clustering and treats each group as a separate component. The value of the K must be provided by the user since MOFBVE has no mechanism to find the optimum number of groups automatically. Secondly, it only calculates the contributions and distributes the resources once at the beginning of the procedure. This means MOFBVE does not update its estimation of contributions during the optimization process. Therefore, the resource allocation schema is fixed and will not be adapted to optimization progress. Thirdly, the amount of resources that MOFBVE allocates to a component is a linear function of its size, not its rank. In other words, the component ranks only determine the order of component selection whereas the size of a component is the main contributor to the portion of the budget it receives.

7.4.4 The New Cooperative Coevolutionary Framework

The New CC Framework (CCFR) is one of the most recent attempts to tackle the imbalance problem [Yang et al. 2017]. Similar to CBCCs, the CCFR consists of exploration and exploitation phases. In the exploration phase, it optimizes all components for exactly one epoch whereas in exploitation phase it continuously optimizes the most contributing component until the contribution of all components converges to the same value. It also adopts a stagnation detection tool that sets the contribution of a component to zero when it detects that the distribution of subsolutions has not been changed for a long time.

Algorithm 5 provides the pseudocode of CCFR. As usual, the first few lines are devoted to initializations and decomposition. Then, in the exploration phase, all components are optimized for one epoch. If the number of generations a component is stagnated reaches to its dimension size, the contribution of that subfunction is reset to zero. This step potentially saves the waste of computational resources on stagnant subpopulations. In each iteration inside the exploitation phase, the most contributing component is selected to be optimized for an extra epoch. Again, if a component is stagnated for a long time, its contribution is reset to zero.

The contribution calculation in CCFR is very similar to CBCC's. In fact, it takes the average of all past contributions and the most recent observed improvement:

$$\bar{\delta}_k^{(t)} = \frac{\bar{\delta}_k^{(t)} + |\overset{*}{f}_p - \overset{*}{f}_c|}{2} \tag{7.8}$$

where $\overset{*}{f}_p$ and $\overset{*}{f}_c$ denote the objective values of the previous and current best solutions (see line 25).

At the very first sight, CCFR and CBCCs look very similar, however, they differ in several ways. Firstly, they use slightly different formulations for calculating contributions (compare Equation (7.2) and Equation (7.8)). Secondly, the exploitation phase in CBCC2 ends as soon as there is no more improvement from the most contributing component (see line 11). In contrast, CCFR only switches from the exploitation phase back to the exploration phase when the estimated contributions of all subfunctions are equal (line 13). Thirdly, CCFR selects the most contributing components inside the exploitation phase (line 14) whereas CBCCs select it before the loop (see line 10). This means it is very likely that CCFR switches between different subfunctions as the index of the most contributing component is updated after each epoch of optimization. Finally, CCFR is armed with a stagnation detection mechanism to avoid spending more resources on the

Algorithm 5 The New Cooperative Coevolutionary Framework

function CCFR(f, D, N, G)
  $(\mathbf{X}, \mathbf{f}) \leftarrow$ initialization(f, N, D) ▷ Initialization
  $(\mathcal{D}, K) \leftarrow$ initGrouping(f) ▷ Decomposition
  $\vec{\mathbf{x}} \leftarrow$ initContextVector$(\mathbf{X}, \mathbf{f})$ ▷ Initial *Context*
  $\bar{\boldsymbol{\delta}} \leftarrow$ zeros$(1, K)$ ▷ Initial contributions
  while termination() $\neq$ True **do** ▷ **Main loop**
    for $k \leftarrow 1 \cdots K$ **do** ▷ **Exploration Phase**
      $\boldsymbol{\eta}[k] \leftarrow 0$
      $(\mathbf{X}, \mathbf{f}, \bar{\boldsymbol{\delta}}, \boldsymbol{\eta}) \leftarrow$ optimize$(\mathbf{X}, \vec{\mathbf{x}}, \mathcal{D}, k, \mathbf{f}, \bar{\boldsymbol{\delta}}, \boldsymbol{\mu})$ ▷ optimization
      $\vec{\mathbf{x}} \leftarrow$ updateContextVector$(\mathbf{X}, \mathbf{f}, \vec{\mathbf{x}})$ ▷ Update *Context*
      if $\boldsymbol{\eta}[k] \geq \mathbf{d}[k]$ **then**
        $\bar{\boldsymbol{\delta}}[k] \leftarrow 0$
    while $\min(\bar{\boldsymbol{\delta}}) \not< \max(\boldsymbol{\mu})$ **do** ▷ **Exploitation Phase**
      $\overset{*}{k} \leftarrow \arg\max \bar{\boldsymbol{\delta}}$ ▷ Select a component
      $(\mathbf{X}, \mathbf{f}, \bar{\boldsymbol{\delta}}, \boldsymbol{\eta}) \leftarrow$ optimize$(\mathbf{X}, \vec{\mathbf{x}}, \mathcal{D}, \overset{*}{k}, \mathbf{f}, \bar{\boldsymbol{\delta}}, \boldsymbol{\eta})$ ▷ 1 epoch optimization
      $\vec{\mathbf{x}} \leftarrow$ updateContextVector$(\mathbf{X}, \mathbf{f}, \vec{\mathbf{x}})$ ▷ Update *Context*
      if $\boldsymbol{\eta}[k] \geq \mathbf{d}[k]$ **then**
        $\bar{\boldsymbol{\delta}}[k] \leftarrow 0$
  return $\mathbf{X}[\arg\min(\mathbf{f}_c), :]$ ▷ Return final solution

function optimize$(f, \mathbf{X}, \vec{\mathbf{x}}, \mathcal{D}, k, \mathbf{f}, \bar{\boldsymbol{\delta}}, \boldsymbol{\eta})$
  for $r \leftarrow 1 \cdots N$ **do** ▷ Replicate $\vec{\mathbf{x}}$ N times
    $\vec{\mathbf{X}}[r, :] \leftarrow \vec{\mathbf{x}}$
  $\vec{\mathbf{X}}[:, \mathcal{D}[k]] \leftarrow \mathbf{X}[:, \mathcal{D}[k]]$ ▷ Plug k^{th} component
  $(\vec{\mathbf{X}}, \mathbf{f}_c) \leftarrow$ optimizer$(f, \vec{\mathbf{X}},)$ ▷ 1 epoch optimization
  $\bar{\boldsymbol{\delta}}[k] \leftarrow (\bar{\boldsymbol{\delta}}[k] + |\min \mathbf{f}_p - \min \mathbf{f}_c|)/2$ ▷ Update contribution
  if isStagnated$(\mathbf{X}[:, \mathcal{D}[k]], \vec{\mathbf{X}}, [:, \mathcal{D}[k]]) =$ True **then**
    $\boldsymbol{\eta}[k] \leftarrow \boldsymbol{\eta}[k] + G$
  else
    $\boldsymbol{\eta}[k] \leftarrow 0$
  $\mathbf{X}[:, \mathcal{D}[k]] \leftarrow \vec{\mathbf{X}}[:, \mathcal{D}[k]]$ ▷ Update solutions
  return $(\mathbf{X}, \mathbf{f}_c, \bar{\boldsymbol{\delta}}, \boldsymbol{\eta})$ ▷ Return to CCFR

function isStagnated$(\mathbf{X}, \vec{\mathbf{X}})$
  for $j \in$ indices$(\mathbf{X})$ **do** ▷ For all variables:
    if mean$(\mathbf{X}[:, j]) \neq$ mean$(\vec{\mathbf{X}}[:, j])$ **then** ▷ Check mean value
      return False
    if std$(\mathbf{X}[:, j]) \neq$ std$(\vec{\mathbf{X}}[:, j])$ **then** ▷ Check std value
      return False
  return True

stagnated components (line 32). This feature is unique to CCFR. The stagnation detection mechanism, however, is very simple. If the mean and variance of all variables of a subfunction of size D_k are not updated for at least D_k iterations, it will be flagged as a stagnated subpopulation. As a result, the contribution of such components reset to zero and will not be considered as the most contributing component regardless of its historical performance (lines 11 and 17). Note that all stagnation counters are reset at the start of the next exploration phase (line 8).

7.5 Chapter Summary

In this chapter, we formally defined the imbalance problem and discussed its effects on the performance of decomposition-based optimization techniques, especially CC algorithms. Then, we reviewed several approaches to address the nonuniform component contribution. Among the reviewed techniques, CBCCs are the simplest models in terms of the number of extra parameters and computational complexity. These algorithms are also easy to understand, implement, and tune.[1] MOFBVE, on the other hand, is the most complex algorithm since it comprises of a number of modules (*e.g.,* sensitivity analysis and clustering algorithms). These elements force computational overhead to the framework and also make it difficult to track the contribution of each module to the overall performance of the algorithm.

The key advantage of modular techniques such as MOFBVE over the others is that it comprises well-known modules which are used and studied in a wide range of scenarios. Therefore, we already know the strengths and weaknesses of each part. However, CBCCs, CCAIO, and CCFR are more *ad hoc* techniques and not designed based on sound theories. Indeed, they are working tools but there exist no theoretical proof or extensive experimental studies to support the proposed heuristics.

An ideal framework to address imbalance problem is an algorithm that is as simple as CBCCs while at the same time it is built based on a more established theory such as bandit-based credit assignment schema. We elaborate these concepts and related techniques in Chapter 10.

[1] In Chapter 8 we study CBCC1 and CBCC2 in more details. Then, we propose CBCC3.

CHAPTER 8

Improving Contribution-Based Cooperative Coevolution

The nonuniformity in the contributions of an optimization problem's components challenges the uniform resource allocation in round-robin Cooperative Coevolution (CC) algorithms. The family of Contribution-Based CC (CBCCs) has been demonstrated a significant improvement over round-robin CCs in solving large-scale imbalanced functions especially when the decomposition is accurate and the level of contribution imbalance is considerably high.

In real-world scenarios, however, we might not have a precise decomposition in hand, or the imbalance level of a function might not be as significant as the benchmarked functions. Therefore, we devote the first part of this chapter to critically analyzing the performance of CBCCs in a more realistic setting. In the second part, we show that over-exploration and over-exploitation are two major sources of performance loss in the CBCCs. On that basis, we propose a new contribution-based algorithm that maintains a better balance between the resources spent on exploration and exploitation phases. The empirical results show that our CBCC3 is superior to its predecessors (*i.e.,* CBCC1 and CBCC2) as well as round-robin CCs.

8.1 Introduction

As discussed in Chapter 2, an effective way to reduce the adverse effect of dimensionality on metaheuristics is inspired by the famous *divide-and-conquer* approach. A widely cited class of algorithms, known as Cooperative Coevolutionary (CC) algorithms [Li 2014], decompose a large-scale complex problem into a set of smaller and simpler subfunction [Omidvar et al. 2014b]. Then, they spend an equal portion of the total computational budget on every subproblem, trying to solve each one of them separately, in a round-robin manner. Although this idea has led to significant improvement over traditional techniques, it ignores an important factor: *imbalance problem* [Omidvar et al. 2015].

In Chapter 7 we formally defined the imbalance problem as the nonuniform contributions of different parts of a partially separable function which can be the result of inequalities in the dimensionality, complexity, and/or importance of different components of a problem [Yang et al. 2017]. Due to these differences, optimizing some subfunctions may have a significant effect on solving the main function, whereas the other components may be only marginally contribute to the overall objective value improvements. Such

imbalance contribution is responsible for significant performance loss because the round-robin CC algorithms distribute the available resources uniformly among all subfunctions regardless of their contributions in solving the main problem [Omidvar et al. 2011].

Several Contribution-Aware CCs have been proposed to enhance the resource allocation scheme in CC framework, by identifying the contributions of components and distribute the resources accordingly. This task can be very challenging as the real-world application are usually black-box problems and hence very little information about the features of subfunctions are available.

Contribution-Based CC (CBCC) is a subclass of Contribution-Aware CCs (CACCs) that allocates the available computational resources to individual components based on their contributions. CBCC variants, namely CBCC1 and CBCC2, have shown significantly better performance than the round-robin CC in a variety of large-scale problems. In general, both CBCC variants keep track of the historical improvements gained from optimizing each subfunction and allocates more resources to the subfunction with the highest accumulated improvements. It has been demonstrated that, given the perfectly accurate decomposition and significantly imbalanced problems, CBCC can improve the standard CC framework [Omidvar et al. 2011]. However, it is still not clear to what extent CBCCs can be effective when employed in a more realistic scenario. Indeed, no one has ever investigated the performance of these techniques when the decomposition is not very accurate, or the imbalance level of subfunctions is relatively low or moderate. Here, we aim to fill this research gap by conducting several numerical experiments to answer the following research questions (RQ):

RQ1: To what extent CBCCs are sensitive to the efficiency of the decomposition?

RQ2: To what extent the imbalance level affects the performance of CBCC? Is it still worthwhile to choose CBCCs over traditional CCs when the level of imbalance is unknown or minor?

RQ3: What are the main limitations of CBCCs and how we can enhance them?

The rest of this chapter is organized as follows. In Section 8.2 we conduct two series of empirical studies to answer RQ1 and RQ2. More precisely, we investigate the effects of decomposition accuracy and imbalance level on the performance of CBCC1 and CBCC2 (see 8.2.1 and 8.2.2). Then, we devote Section 8.3 to enhancing the limitations of CBCCs (see RQ3). In the first part of this section, we conduct a brief case study to identify the common shortcomings of CBCCs. Next, we propose a new algorithm called CBCC3 that improves the resource allocation schema of CBCC framework in Subsection 8.3.2. To show the potentials of CBCC3, we perform a set of experiments in Subsection 8.3.3 and discuss the results in Subsection 8.3.3. Finally, we conclude this chapter in Section 8.4 by summarizing the findings and rising some important research questions that we will address in the following chapters.

8.2 Sensitivity Analyses

To answer RQ1 and RQ2 in Section 8.1, we design two sets of experiments to: i) analyze the sensitivity of CBCC1 and CBCC2 to the accuracy of the given decomposition, and ii) examine the sensitivity of CBCCs to the level of contribution imbalance. Subsections 8.2.1 and 8.2.2 provide more information on these experiments and their results. Both series of experiments share some common parameters and designs that are provided in Subsection 8.2.

Table 81: Parameters and their values for the sensitivity analysis in Section 8.2

Parameter	Description	Value
N	population size	50
D	dimensionality of the main problems	1000
δ_t	epoch length	10
v	CBCC variant	either 1 or 2
$termination()$	termination criterion	function calls $\geq 3e{+}06$

Experiments Design

We study the performance of CBCC1 and CBCC2 through aforementioned experiments. To have a reference for comparisons, we also run all experiments on a well-known variation of non-contribution-aware CC algorithm [Yang et al. 2007a]. As it is very common in the literature of large-scale optimization, we adopt SaNSDE [Yang et al. 2008c] as the subproblem optimizer. All experiments are executed for 25 independent runs. Table 81 provides the parameters and their values. More details about implementations of CC and CBCC algorithms are available in Chapter 7.

Metrics and Measures

Before presenting the experiments and analyses, we need to define the metrics we will use to compare CBCC variants: *self-improvement*, *relative improvement*, *statistical difference* and *win-draw-lose* numbers.

Self-improvement: In this section, self-improvement indicates to what degree an algorithm improves its performance in comparison with itself in different situations. For example, self-improvement for CBCC1 in the first set of experiments shows how much its performance varies when decomposition accuracy changes. More formally:

$$P_{self}(i, j : \mu) = \frac{\log \bar{f}(i, j : 0) - \log \bar{f}(i, j : \mu)}{\log \bar{f}(i, j : 0)} \times 100 \tag{8.1}$$

where $P(i, j : \mu)$ is the self-improvement of algorithm i on function j. In (Equation (8.1)), $\bar{f}$ is the median of fitness values of final solutions obtained from 25 independent runs. The parameter μ indicates either the percentage of noisy variables (ranging from 0% to 50%) or imbalance level (ranging from 0 to 3) in different experiments. Therefore, value 0 in $\bar{f}(i, j : 0)$ means either an errorless decomposition (in Subsection 8.2.1) or a balanced problem (in Subsection 8.2.2).

Relative improvement: In contrast with self-improvement, the relative improvement means the magnitude of enhancement that an algorithm gains in comparison with the control method in similar settings. More formally:

$$P_{relative}(i, j : \mu) = \frac{\log \bar{f}(c, j : \mu) - \log \bar{f}(i, j : \mu)}{\log \bar{f}(c, j : \mu)} \times 100 \tag{8.2}$$

where c indicates the control method which is CC in all experiments (see [Yang et al. 2007a]).

By definition, both improvement measures can take negative values which indicate a drop in efficiency.

Statistical difference: The concept of statistical difference may be reflected in various phrases, such as significantly different results or statistically similar performances. Following what is common in technical publications, by significantly different results we mean

not only the basic statistics (*i.e.,* mean and median) are very different, but also the distributions of the obtained results have no or negligible overlaps. In contrast, statistically similar performances implies that the results are very likely to be drawn from the same statistical distribution. Note that, technically, two algorithms perform either statistically similar or significantly different regarding the employed significance test.

There are many ways to determine whether two algorithms perform statistically similar or different [Derrac et al. 2011]. In this study, we use Kruskal-Wallis rank-sum test to obtain the p-values, and Wilcoxon rank sum test for pair-wise comparison [García et al. 2010]. Both techniques are nonparametric and mainly compare the medians of the samples. Therefore, in the tables showing basic statistics, the medians of significantly different results (in comparison with CC as the control method) are marked as **bold**. Median values with regular font indicate the algorithms with similar performance to CC's.

Win-draw-lose: To compare each variation of CBCCs with CC as the baseline, win-draw-loss procedure is adopted. The three numbers separated by dashes respectively indicates the number of times the testing algorithm wins, ties or loses against the control method. Note that the results should be significantly different to be counted as a win or lose. Thus, all statistically similar results are counted as draws.

8.2.1 Sensitivity to Decomposition Accuracy

In this part, we examine the sensitivity of two variations of CBCC to the decomposition efficiency. There are many ways to add some uncertainty or error in order to artificially decrease the decomposition accuracy of ideal grouping [Omidvar et al. 2011] from 100% towards the desired level. For example, one can divide a nonseparable component into further subcomponents, group separable components to make a larger group or even swap variables across different components [Chen and Tang 2013].

In this study, we introduce another type of decomposition error. We assume that in a practical situation, the adopted decomposition technique cannot detect all variable interactions. Therefore, it groups all unclassified variables into an extra group that we call *unlabeled group* (*i.e.,* the group of all unclassified variables). Obviously, in a perfect setting, an ideal decomposition algorithm groups all variables correctly, and hence, the size of unlabeled group would be zero. In a realistic setting, the percentage of unclassified variables, and thus the size of the unlabeled group, may increase due to the challenges in problem decomposition and the efficiency of decomposition technique.

In this study, we examine various decomposition error levels (may also be referred as noise percentage) ranging from 0% (*i.e.,* ideal decomposition) to 50% of all variables (*i.e.,* poor decomposition). To do so, we randomly select a percentage of variables uniformly from all ideally decomposed components and aggregate them into the unlabeled group. The resulting component contains variables from all other groups. Consequently, this group has strong interactions with all other components.

Results and Discussions

Table 82 presents the basic statistics (*i.e.,* median, mean and standard deviation) of CC and CBCCs with various decomposition accuracies. The **bold** numbers indicate the significantly different results of CBCCs when compared with CC as the baseline technique. For a deeper investigation, the self and relative improvement measures are depicted in Figure 81 and 82, respectively.

As Figure 81a reveals, decomposition error has an adverse effect on the performance of CC on almost all functions. Similar trends are observable in Figure 81b and Figure 81c

for CBCC1 and CBCC2. These results were expected because the decomposition error (especially the type that we introduce in this study) increases the interaction between components. Therefore, solving a particular component is not independent of the other components as it was in the case of ideal decomposition. Another effect of this type of decomposition error is that the size of the new group (the unlabeled component) grows by increasing the error rate. For example, when error level reaches 50%, the dimensionality of the unlabeled component reaches 500, while some of the components have less than 15 variables. This means the added error magnifies the imbalance level in terms of components dimensional sizes.

A surprising observation from Figures 81a–81b is that by decreasing the decomposition accuracy, the performance of CC and CBCC1 are improved on a few functions, including f_6. This is probably due to an indirect effect of decomposition error on the imbalance level existing in the original problem. When a considerable number of variables are moved from the most contributing component to the unlabeled group (*i.e.,* the recipient component), the contribution of the donor component drops dramatically (*i.e.,* imbalance level falls). In other words, the type of decomposition error which we introduce in this study can spread the contributing variables and occasionally convert a highly imbalanced problem into a more balanced problem. Therefore, we may observe marginal improvements in some cases.

Table 82: Comparing CC and CBCCs with different levels of decomposition error on f_4–f_{11} from CEC'2013 LSGO benchmark.

		5% error			10% error			20% error			30% error			50% error		
		CC	**CBCC1**	**CBCC2**	**CC**	**CBCC1**	**CBCC2**	**CC**	**CBCC1**	**CBCC2**	**CC**	**CBCC1**	**CBCC2**	**CC**	**CBCC1**	**CBCC2**
f_4	m	5.44e+08	2.84e+08	**6.97e+08**	7.47e+08	2.87e+09	**1.84e+09**	5.42e+09	5.45e+09	**2.03e+10**	1.15e+10	1.12e+10	**2.50e+10**	3.24e+10	3.34e+10	**5.54e+10**
	μ	1.38e+09	1.40e+09	2.56e+09	2.27e+09	4.38e+09	5.73e+09	8.41e+09	8.41e+09	2.40e+10	1.24e+10	1.22e+10	3.73e+10	3.29e+10	2.83e+10	8.37e+10
	σ	2.13e+09	2.13e+09	3.60e+09	2.96e+09	4.50e+09	6.79e+09	7.34e+09	7.80e+09	2.42e+10	7.70e+09	7.65e+09	2.88e+10	1.71e+10	1.49e+10	1.09e+11
f_5	m	2.90e+06	**2.48e+06**	**2.54e+06**	2.87e+06	2.49e+06	2.37e+06	2.98e+06	2.74e+06	3.02e+06	3.04e+06	3.00e+06	3.55e+06	4.33e+06	3.96e+06	4.47e+06
	μ	2.94e+06	2.51e+06	2.53e+06	2.81e+06	2.60e+06	2.54e+06	2.86e+06	2.81e+06	3.03e+06	3.12e+06	3.25e+06	3.55e+06	4.26e+06	4.04e+06	4.47e+06
	σ	4.47e+05	4.70e+05	5.36e+05	5.59e+05	5.18e+05	5.21e+05	4.15e+05	4.43e+05	7.36e+05	6.68e+05	9.03e+05	7.73e+05	6.66e+05	9.49e+05	8.50e+05
f_6	m	8.35e+04	7.94e+04	**1.01e+05**	8.55e+04	7.73e+04	**1.16e+05**	7.51e+04	**8.37e+00**	**1.12e+05**	2.16e+01	3.64e+01	**1.02e+05**	3.94e+01	**5.41e+03**	**9.86e+04**
	μ	8.19e+04	7.88e+04	1.06e+05	8.46e+04	6.99e+04	1.21e+05	6.56e+04	2.40e+04	1.03e+05	1.69e+04	2.00e+04	9.62e+04	9.36e+02	3.38e+04	9.93e+04
	σ	2.36e+04	3.11e+04	1.76e+04	1.70e+04	2.98e+04	2.51e+04	3.28e+04	3.14e+04	2.30e+04	2.78e+04	2.77e+04	2.37e+04	2.06e+03	3.94e+04	1.69e+04
f_7	m	5.89e+07	7.39e+07	**9.12e+07**	4.92e+07	4.62e+07	**1.29e+08**	4.62e+07	5.28e+07	**1.40e+08**	4.53e+07	7.97e+07	**3.49e+08**	7.38e+07	7.89e+07	**1.31e+09**
	μ	6.25e+07	9.37e+07	1.18e+08	5.69e+07	4.97e+07	1.95e+08	5.65e+07	7.95e+07	3.05e+08	7.75e+07	1.84e+08	5.24e+08	1.08e+08	2.23e+08	1.65e+09
	σ	5.26e+07	7.81e+07	1.10e+08	5.82e+07	2.85e+07	2.08e+08	4.89e+07	1.37e+08	3.70e+08	1.33e+08	3.00e+08	5.14e+08	1.26e+08	4.69e+08	1.02e+09
f_8	m	1.38e+14	**4.39e+13**	**3.23e+13**	1.52e+14	1.31e+14	1.36e+14	3.81e+14	3.01e+14	4.67e+14	4.49e+14	4.00e+14	6.41e+14	5.11e+14	4.18e+14	**8.42e+14**
	μ	1.47e+14	7.57e+13	1.34e+14	2.05e+14	2.26e+14	3.20e+14	4.93e+14	3.92e+14	3.50e+15	6.02e+14	5.01e+14	2.19e+15	6.31e+14	6.15e+14	2.35e+15
	σ	1.04e+14	1.03e+14	2.14e+14	2.21e+14	2.95e+14	4.90e+14	3.46e+14	3.03e+14	1.04e+16	4.11e+14	4.20e+14	6.14e+15	3.50e+14	4.51e+14	4.85e+15
f_9	m	3.22e+08	**2.40e+08**	**2.45e+08**	3.16e+08	**2.43e+08**	**2.39e+08**	2.95e+08	**2.35e+08**	**2.47e+08**	3.41e+08	**2.77e+08**	3.08e+08	3.96e+08	4.11e+08	3.90e+08
	μ	3.17e+08	2.33e+08	2.39e+08	3.10e+08	2.41e+08	2.36e+08	3.03e+08	2.37e+08	2.56e+08	3.40e+08	2.84e+08	3.26e+08	3.81e+08	4.10e+08	3.95e+08
	σ	4.15e+07	4.10e+07	4.51e+07	4.04e+07	3.28e+07	4.23e+07	4.20e+07	4.55e+07	5.32e+07	4.46e+07	6.84e+07	6.07e+07	6.80e+07	5.68e+07	6.68e+07
f_{10}	m	5.96e+01	6.53e+01	6.02e+01	6.03e+01	5.16e+01	6.72e+01	1.65e+02	1.38e+02	**4.59e+06**	3.26e+02	3.20e+02	**5.67e+06**	1.34e+03	1.25e+03	**9.28e+04**
	μ	5.87e+01	6.09e+01	8.20e+05	5.03e+01	5.28e+01	8.57e+05	2.72e+02	1.60e+02	4.30e+06	3.94e+02	3.54e+02	3.77e+06	2.64e+05	1.03e+06	3.34e+06
	σ	1.52e+01	1.82e+01	1.93e+06	2.41e+01	1.69e+01	2.03e+06	4.70e+02	9.86e+01	3.32e+06	1.82e+02	1.54e+02	3.54e+06	1.31e+06	2.46e+06	3.69e+06
f_{11}	m	9.68e+08	8.82e+08	**4.70e+09**	2.22e+09	1.52e+09	1.17e+10	2.52e+09	5.02e+09	**1.46e+10**	7.79e+09	4.99e+09	**4.89e+10**	2.35e+10	2.85e+10	**1.53e+11**
	μ	8.12e+09	8.10e+09	2.44e+10	2.00e+10	1.41e+10	2.88e+10	2.37e+10	2.40e+10	4.94e+10	1.97e+10	1.92e+10	8.54e+10	3.96e+10	3.93e+10	2.44e+11
	σ	1.84e+10	1.81e+10	3.43e+10	3.54e+10	2.23e+10	3.61e+10	3.90e+10	3.77e+10	8.90e+10	2.73e+10	3.64e+10	1.40e+11	3.68e+10	2.91e+10	2.55e+11

Table 83: Win-Draw-Lose analysis of CBCC1 and CBCC2 on f_4–f_{11} (CEC'13 LSGO) with different decomposition accuracy. In all cases, the round-robin CC is used as the baseline.

	Error					
	0%	5%	10%	20%	30%	50%
CBCC1	4-4-0	3-5-0	1-7-0	2-6-0	1-7-0	0-7-1
CBCC2	2-4-2	3-1-4	1-4-3	1-2-5	0-3-5	0-2-6

Comparing the performance of CBCC2 in Figure 81c with the other two algorithms reveals that this technique is more sensitive to the decomposition accuracy. Indeed, CBCC2 not only has negative improvements with the largest magnitudes, but also it shows no exceptional improvement by increasing error (as we observed in the other techniques on f_6).

Figure 81 depicts the sensitivity of each algorithm to the decomposition accuracy. Nevertheless, it does not show which algorithm is the most promising one when the decomposition is imperfect. In other words, we need to investigate whether CBCCs have any chance to improve CC at the presence of decomposition error. Figure 82 clearly illustrates these comparisons. It illustrates the relative improvement each variation of CBCC achieves in comparison with CC as the control method (see Equation (8.2)) for more information).

As Figure 82a demonstrates, CBCC1 improves CC in most cases. The magnitudes of improvements, however, drop as the accuracy decreases. Among all functions, f_6 shows an abnormal behavior. Indeed, CBCC1 performs relatively poor when the decomposition efficiency drops to 50%. The reason behind this phenomenon is the good performance of CC on this function (see Figure 81a). As mentioned earlier, the added error distributes a large number of variables from the most contributing component to the new group. This sometimes results in an imbalance reduction which is in favor of round-robin CCs.

As it was expected, CBCC2 performs relatively poor in the presence of decomposition error. Although its performance is comparable to CC using ideal grouping, its efficiency significantly deteriorates when the error level reaches 10% or goes beyond. The greedy nature of CBCC2 seems to be the main driving factor for this phenomenon: It spends an unnecessarily large portion of the budget to optimize the component with the highest record of improvement at the early exploitation phases, whilst it should frequently switch between promising component and unlabeled group.

Table 83 concludes this part of experiments. As the win-draw-lose numbers confirm, CBCC1 performs better or at least similar to CC in all cases, except when error reaches 50%. In contrast, the performance of CBCC2 is always worse than CC except for the ideal decomposition. As a result, one can adopt CBCC1 without the risk of degrading the CC's performance. Contrarily, CBCC2 may be used only when the practitioners are confident about their decomposition accuracy.

8.2.2 Sensitivity to Imbalance Level

In this set of experiments, we investigate the sensitivity of CBCC algorithms to the imbalance level. There are several ways of introducing imbalance to partially separable problems. For example, in [Omidvar et al. 2015] imbalance is added to the subfunctions

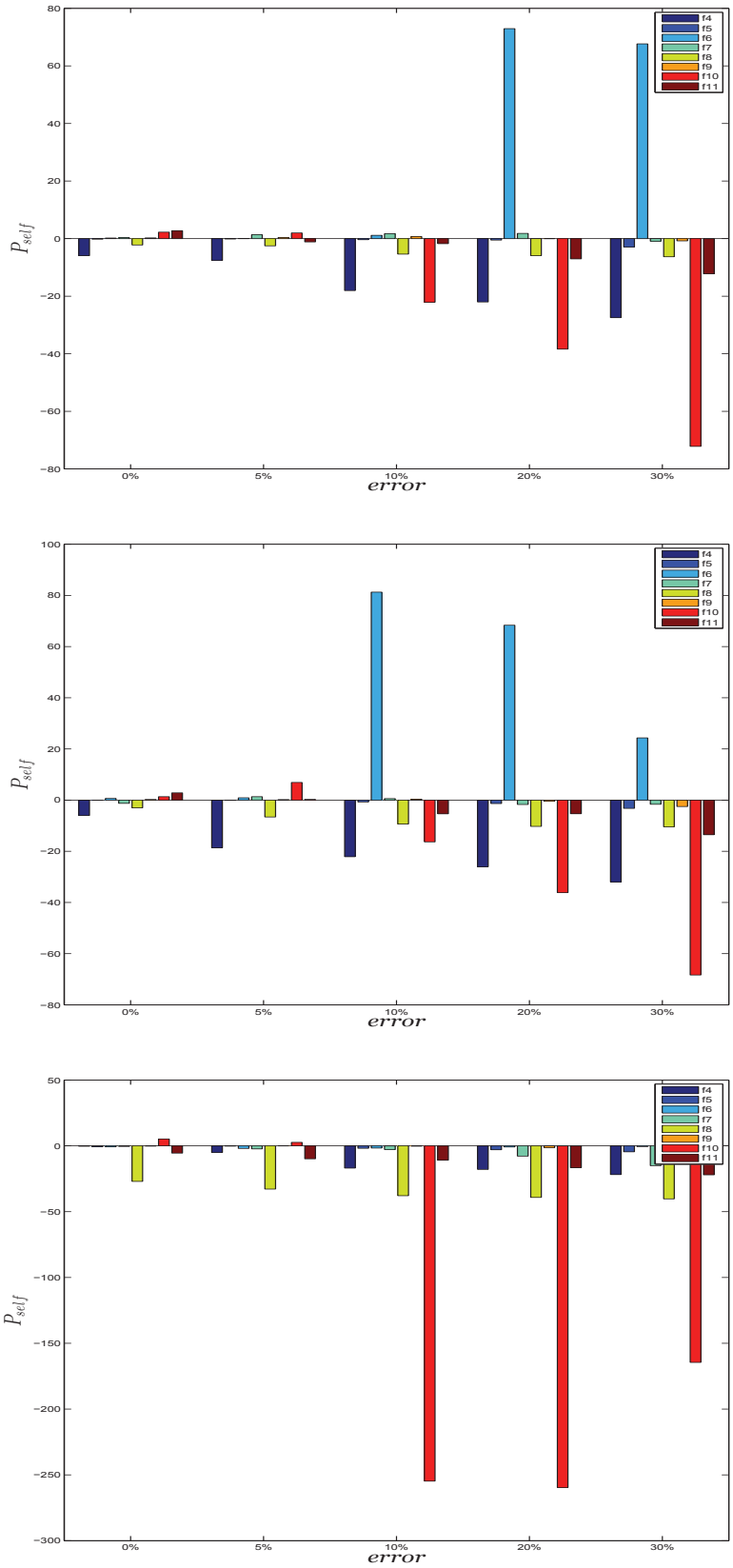
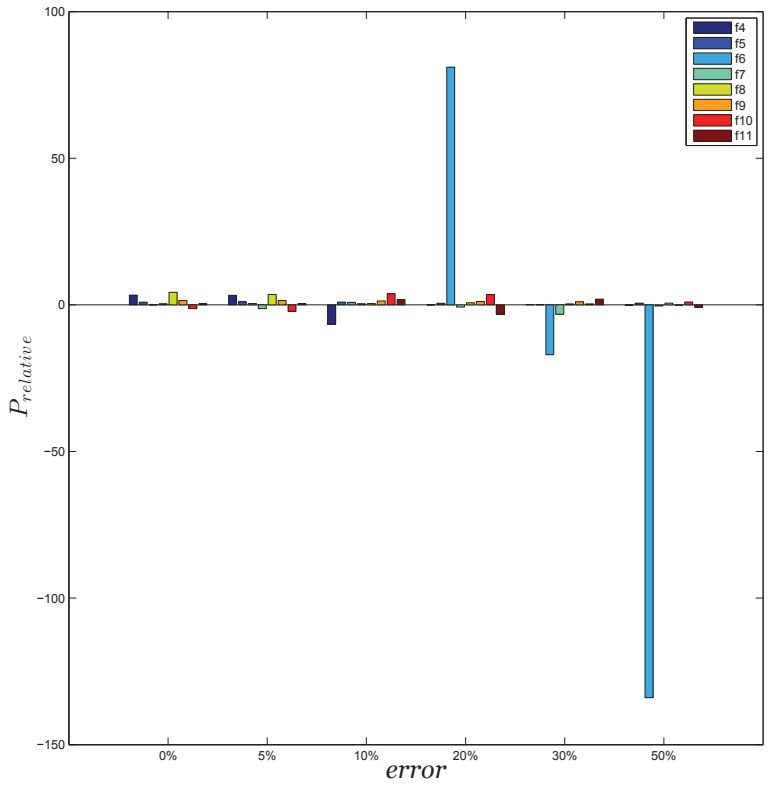
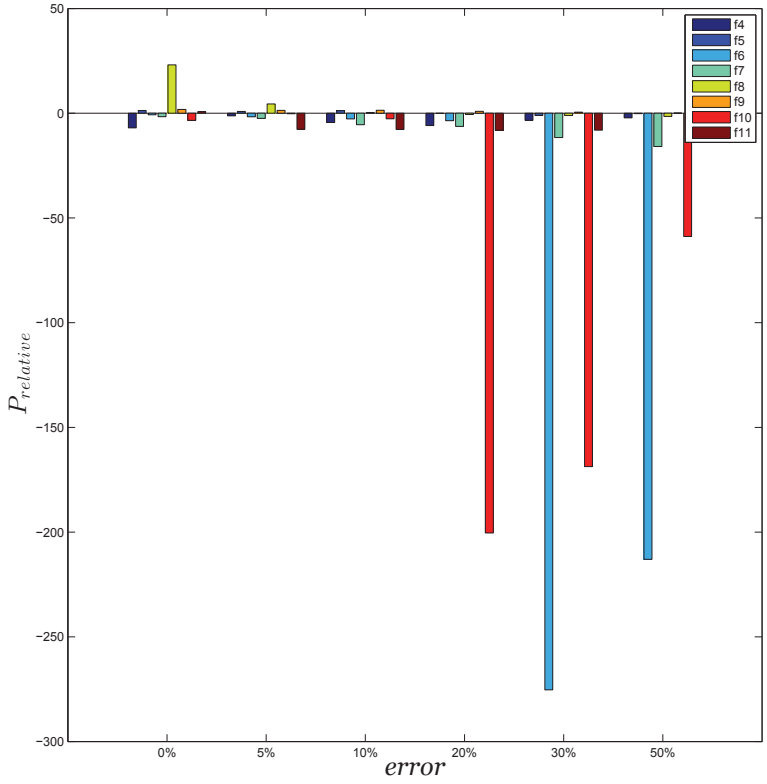


Figure 81: Sensitivity to decomposition accuracy: Self-improvement of CC, CBCC1 and CBCC2 on f_4 to f_{11} from the CEC'13 LSGO benchmark (see Equation (8.1)).



(a) CBCC1



(b) CBCC2

Figure 82: Sensitivity to decomposition accuracy: Relative improvement of CBCC1 and CBCC2 on f_4 to f_{11} from the CEC'13 LSGO benchmark (see Equation (8.2)).

using nonuniform coeffiecients. More formally, the imbalanced problems are defined as follows:

$$f(\mathbf{x}) = \sum_{k=1}^{K} C_k . f_k(\mathbf{x}_k) \tag{8.3}$$

where $\mathbf{x}_k$ is the kth group of variables and each C_k is a coefficient randomly chosen from the following distribution:

$$C_k \sim 10^{S\mathcal{N}(0,1)} \tag{8.4}$$

In Equation (8.4), $\mathcal{N}(0,1)$ is a Gaussian distribution with zero mean and unit variance. In CEC'13 LSGO benchmark test suite S is fixed to 3 [Li et al. 2013a]. Another source of imbalance in this suite is unequal sizes of subfunction (*i.e.,* $\exists k, \exists \grave{k}$ s.t. $|\mathbf{x}_k| \neq |\mathbf{x}_{\grave{k}}|$).

In this study, we use partially separable functions from CEC'13 LSGO benchmark suite (*i.e.,* f_4 to f_{11}). To control the imbalance level, we vary the value of parameter S in Equation (8.4). More precisely, we examine the performance of CBCC techniques on functions with $S \in \{0, 1, 2, 3\}$. Consequently, we have four different levels of imbalanced functions ranging from almost balanced[1] to minor, moderate and strongly imbalanced functions. Other factors such as dimension sizes and number of components are not modified.

Results and Discussions

Table 84 provides the basic statistics of CC, along with CBCC1 and CBCC2 on functions with different levels of imbalance. Similar to the previous table, the bold values demonstrate the significantly different results when compared with the baseline (*i.e.,* round-robin CC).

We do not draw bar charts for this set of experiments because various values for S (in Equation (8.4)) will result in significantly different coefficients (C_k). As a direct consequence, the objective value of a particular solution $\mathbf{x}$ is greatly changed when the level of imbalance is varied (see Equation (8.3)). Therefore, the problems with different imbalance levels (although having the same basis functions and dimensionality) can be considered as totally different functions. This makes the concept of self-improvement unmeasurable in this context.

Figure 83 illustrates the relative improvements that CBCCs achieved over eight functions with four different variations in coefficients distribution. As Figure 83a confirms, CBCC1 is superior to CC in almost all functions regardless of the level of imbalance. In very rare occasions, CBCC1 fails to improve the traditional CC. However, the number and magnitude of positive improvements outweigh those few failed cases.

CBCC2, in contrast with CBCC1, demonstrates a higher sensitivity to the imbalance level. As Figure 83b illustrates, in balanced problems (*i.e.,* $C_k = 1, \forall C_k$) CBCC2 performs poorly in comparison with CC. In imbalanced problems, however, CBCC2 presents a better performance even though its improvement over CC is not very significant.

Finally, Table 85 concludes this section. Overall, CBCC1 performs significantly better than CBCC2, regardless of the level of imbalance. As expected, both variations of CBCC perform more efficiently when the imbalance level is considerably significant. However, since CBCC1 is never significantly worse than CC, we recommend adopting it instead of round-robin CCs, regardless of the imbalance severity.

[1] Even for $S = 0$, the subfunctions are not completely balanced due to the variations in their dimensionalities.

Table 84: Experimental results of CC and variants of CBCC on CEC'2013 benchmark with different levels of imbalance.

		$C_k \sim 10^{0\mathcal{N}(0,1)}$			$C_k \sim 10^{1\mathcal{N}(0,1)}$			$C_k \sim 10^{2\mathcal{N}(0,1)}$			$C_k \sim 10^{3\mathcal{N}(0,1)}$		
		CC	**CBCC1**	**CBCC2**	**CC**	**CBCC1**	**CBCC2**	**CC**	**CBCC1**	**CBCC2**	**CC**	**CBCC1**	**CBCC2**
f_4	Median	3.57e+07	**1.11e+07**	**5.20e+09**	2.78e+07	**7.16e+06**	**9.91e+09**	4.78e+07	4.65e+07	**5.89e+08**	1.76e+08	**9.37e+07**	**6.59e+08**
	Mean	3.74e+07	2.06e+07	5.37e+09	3.34e+07	1.06e+07	1.10e+10	4.89e+07	6.06e+07	5.88e+08	1.96e+08	1.05e+08	6.50e+08
	StDev	1.79e+07	4.06e+07	1.70e+09	1.85e+07	1.10e+07	4.42e+09	1.71e+07	3.84e+07	1.53e+08	1.08e+08	5.01e+07	1.83e+08
f_5	Median	8.25e+03	7.83e+03	8.98e+03	1.18e+04	1.13e+04	**1.35e+04**	1.09e+05	9.62e+04	1.06e+05	2.82e+06	**2.46e+06**	**2.31e+06**
	Mean	8.39e+03	7.89e+03	8.93e+03	1.19e+04	1.13e+04	1.43e+04	1.09e+05	9.88e+04	1.01e+05	2.76e+06	2.43e+06	2.35e+06
	StDev	9.80e+02	1.16e+03	1.26e+03	9.68e+02	9.06e+02	3.64e+03	2.04e+04	1.31e+04	1.43e+04	6.18e+05	3.13e+05	3.88e+05
f_6	Median	1.41e+01	1.37e+01	1.30e+01	8.16e+01	8.32e+01	7.97e+01	2.50e+03	2.39e+03	2.23e+03	8.55e+04	8.55e+04	9.29e+04
	Mean	1.39e+01	1.38e+01	1.35e+01	8.24e+01	8.30e+01	8.03e+01	2.44e+03	2.48e+03	2.23e+03	8.96e+04	8.73e+04	9.63e+04
	StDev	1.34e+00	1.59e+00	1.75e+00	1.89e+01	1.49e+01	1.22e+01	5.10e+02	5.31e+02	8.88e+02	2.08e+04	3.03e+04	1.98e+04
f_7	Median	6.22e+05	6.31e+05	5.75e+05	1.43e+06	1.52e+06	1.52e+06	9.53e+06	5.77e+06	1.08e+07	6.27e+07	5.87e+07	8.45e+07
	Mean	5.99e+05	5.87e+05	6.15e+05	1.50e+06	1.53e+06	1.67e+06	9.35e+06	6.46e+06	9.78e+06	6.12e+07	7.46e+07	1.29e+08
	StDev	2.36e+05	2.15e+05	1.62e+05	6.79e+05	5.31e+05	8.74e+05	4.37e+06	5.53e+06	5.92e+06	3.58e+07	5.52e+07	2.44e+08
f_8	Median	7.25e+07	6.99e+07	**9.40e+09**	2.45e+08	**2.06e+08**	**4.95e+08**	7.81e+10	**1.41e+10**	**3.85e+09**	6.85e+13	**1.74e+13**	**4.35e+10**
	Mean	7.18e+07	6.96e+07	9.52e+09	2.55e+08	2.04e+08	4.94e+08	8.28e+10	1.98e+10	3.74e+09	7.58e+13	2.61e+13	1.28e+14
	StDev	6.32e+06	1.08e+07	4.60e+09	6.94e+07	3.73e+07	9.43e+07	4.04e+10	1.43e+10	6.48e+08	5.05e+13	2.99e+13	6.41e+14
f_9	Median	4.67e+03	4.58e+03	4.73e+03	6.03e+04	**5.38e+04**	**5.44e+04**	3.65e+06	**3.01e+06**	**3.02e+06**	3.39e+08	**2.52e+08**	**2.37e+08**
	Mean	4.55e+03	4.58e+03	5.04e+03	5.82e+04	5.27e+04	5.33e+04	3.41e+06	3.06e+06	2.92e+06	3.21e+08	2.38e+08	2.33e+08
	StDev	3.95e+02	3.08e+02	1.04e+03	7.07e+03	5.61e+03	1.00e+04	5.89e+05	3.51e+05	4.76e+05	6.00e+07	4.49e+07	4.74e+07
f_{10}	Median	9.43e+00	9.16e+00	9.68e+00	1.43e+01	1.40e+01	1.42e+01	2.87e+01	2.99e+01	3.03e+01	6.55e+01	6.91e+01	7.55e+01
	Mean	9.16e+00	9.13e+00	9.39e+00	1.38e+01	1.41e+01	1.43e+01	3.03e+01	3.01e+01	2.98e+01	7.06e+01	7.20e+01	7.14e+01
	StDev	1.15e+00	1.47e+00	1.09e+00	1.83e+00	2.16e+00	2.20e+00	5.67e+00	5.03e+00	5.24e+00	1.51e+01	1.63e+01	1.27e+01
f_{11}	Median	3.42e+06	3.50e+06	3.82e+06	1.23e+07	1.18e+07	1.31e+07	1.25e+08	1.09e+08	1.02e+08	1.74e+09	1.61e+09	1.47e+09
	Mean	3.49e+06	3.56e+06	3.93e+06	1.22e+07	1.15e+07	1.34e+07	1.65e+08	1.54e+08	1.47e+08	2.14e+10	1.33e+10	4.79e+09
	StDev	6.35e+05	6.95e+05	1.02e+06	2.68e+06	3.15e+06	3.21e+06	1.33e+08	1.17e+08	1.54e+08	3.09e+10	2.51e+10	1.19e+10

Table 85: Win-Draw-Lose analysis of CBCC1 and CBCC2 on f_4–f_{11} (CEC'13 LSGO) with different levels of imbalance. The values of S is used in $C_k \sim 10^{S\mathcal{N}(0,1)}$ to define the imbalance severity.

	S			
	1	2	3	4
CBCC1	1-7-0	3-5-0	2-6-0	4-4-0
CBCC2	0-6-2	1-2-5	2-5-1	2-4-2

8.3 Improving Resource Allocation in CBCC

In the previous section we investigated the performance of CBCCs in the face of decomposition error as well as various levels of imbalance. These studies confirm the potentials of CBCC framework. However, CBCCs can still be improved. In this section, we conduct a number of case studies to identify their pitfalls and then enhance them.

8.3.1 Case Studies

Similar to the previous experiments in Section 8.2 we limit the experiments to the partially separable functions of the CEC'13 LSGO benchmark suite which exhibit some degrees of contribution imbalance (*i.e.,* f_4-f_{11}). For the sake of simplicity, we use ideal decomposition for all experiments and, unlike the previous experiments in this chapter, we do not modify the contribution imbalance (*e.g.,* C_k values).

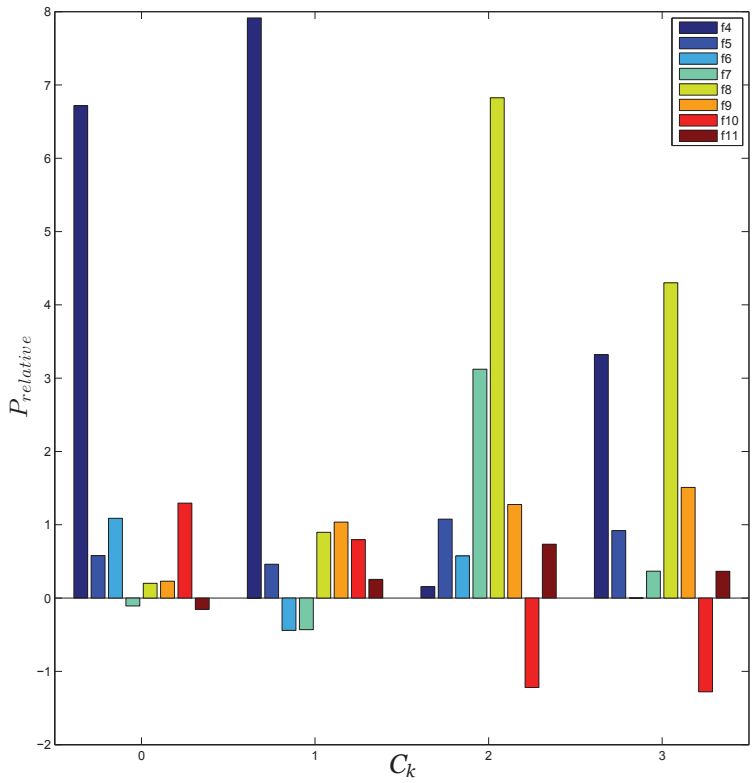
Table 86 contains the median, mean, and the standard deviation of 25 independent runs for CC, CBCC1 and CBCC2. We used Wilcoxon rank-sum [Derrac et al. 2011] test to compare CBCCs against the baseline. The **bold** entries in the table show that CBCC significantly outperforms CC with a 95% confidence interval. The entries that are marked with the symbol '≈' indicate the performance of either version of CBCC is statically similar to the baseline. In the cases that the round-robin CC significantly outperforms either of the CBCCs, we print '↓' next to the entry with the worse performance.

Table 86 suggests that both CBCCs perform statistically similar to CC on most functions (62.5%) and only perform significantly better on three out of eight problems. Comparing this with the success rate of CBCCs on the CEC'10 LSGO benchmarks (as reported in [Omidvar et al. 2011]) we can observe that their performance significantly dropped from 60% on CEC'10 LSGO to 37.5% in CEC'13 LSGO benchmarks.

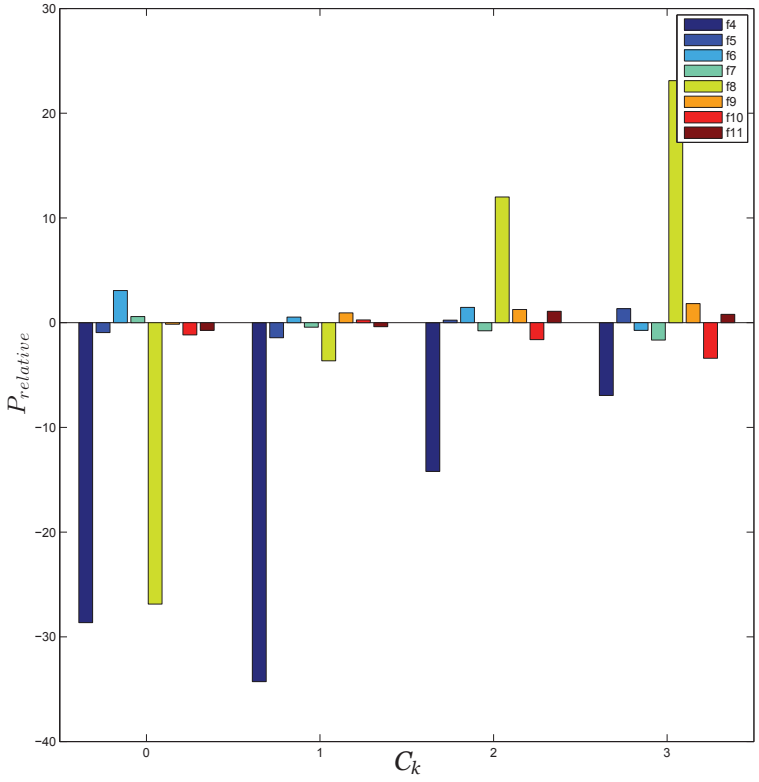
For a better understanding of the root cause of this performance drop, we plot the portion of the computational budget each component received in Figure 84. The horizontal axis indicates the coefficients associated with each subfunction in a logarithmic scale. The vertical axis shows the total number of evaluations (*i.e.,* the length of epochs × the number of epochs) allocated to each component. Note that the nonuniformity in component coefficients is not the only source of contribution imbalance since other factors such as unequal component sizes and heterogeneous landscapes can also contribute. Nevertheless, considering the differences between the magnitudes of the coefficients, we can conclude that the nonuniform coefficients have the strongest relationship with the variations in components' contributions.

The main observations from Figure 84 are:

1. Both CBCC variants accurately identified the component with the largest coefficient



(a) CBCC1



(b) CBCC2

Figure 83: Sensitivity to imbalance level: Relative improvement of CBCC1 and CBCC2 on f_4 to f_{11} from the CEC'13 LSGO benchmark (see Equation (8.2)).

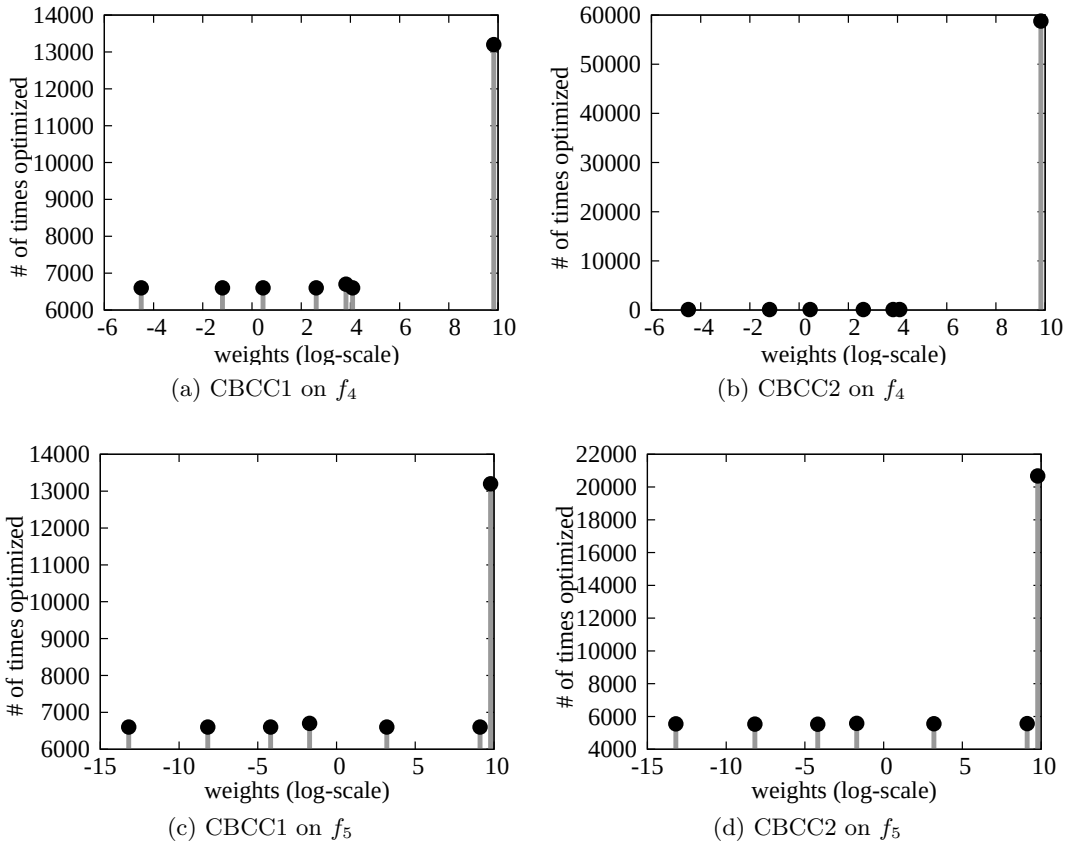
Table 86: Comparison between CC and CBCC1 and CBCC2 on f_4–f_{11} (CEC'13 LSGO). Highlighted entries are significantly better than the baseline (CC) according to Wilcoxon rank-sum test ($\alpha = 0.05$).

		CC	CBCC1	CBCC2
f_4	m	1.53e+08	**6.54e+07**	9.03e+10$^{\downarrow}$
	μ	1.97e+08	7.71e+07	8.77e+10
	σ	1.51e+08	4.05e+07	1.14e+10
f_5	m	2.65e+06	2.29e+06$^{\approx}$	**2.06e+06**
	μ	2.66e+06	2.28e+06	2.09e+06
	σ	7.12e+05	3.55e+05	3.52e+05
f_6	m	8.74e+04	8.74e+04$^{\approx}$	8.35e+04$^{\approx}$
	μ	8.57e+04	8.85e+04	8.39e+04
	σ	1.95e+04	2.88e+04	2.36e+04
f_7	m	4.53e+07	6.23e+07$^{\approx}$	7.85e+07$^{\approx}$
	μ	5.12e+07	6.38e+07	8.82e+07
	σ	3.67e+07	4.01e+07	6.78e+07
f_8	m	5.43e+13	**1.09e+13**	**1.90e+12**
	μ	7.19e+13	1.38e+13	1.88e+12
	σ	6.07e+13	1.14e+13	2.80e+11
f_9	m	2.95e+08	**2.34e+08**	**2.00e+08**
	μ	2.85e+08	2.32e+08	2.03e+08
	σ	6.20e+07	4.85e+07	2.45e+07
f_{10}	m	7.05e+01	7.51e+01$^{\approx}$	7.17e+01$^{\approx}$
	μ	6.90e+01	7.44e+01	7.16e+01
	σ	1.68e+01	9.97e+00	1.36e+01
f_{11}	m	1.51e+10	1.41e+09$^{\approx}$	1.44e+09$^{\approx}$
	μ	2.62e+10	1.58e+10	1.63e+10
	σ	3.10e+10	2.26e+10	2.76e+10

and repeatedly selected it for further optimization in the exploitation phase. This can be an effective strategy as long as the chosen component maintains its contribution during optimization. Later, we will see that this is not necessarily the case as in several scenarios the CBCCs (and in particular CBCC2) fail to switch to another component that just turned to be the new most contributing component.

2. Due to the dominance of one component in terms of contribution to the overall solution quality, the equal allocation of resources through the exploration phase wastes a considerable amount of computational budget. This can be the main reason that CBCC1 cannot outperform CC in f_5 (see Table 86), although it spent more resources on the optimization of the component with the largest coefficient (see Figure 84).

To investigate whether the first observation is always the best strategy, let us consider the CBCC2 performance on f_4. Although CBCC2 spent the largest portion of the budget on the component with the largest coefficient, its overall performance is significantly worse



(a) CBCC1 on f_4 (b) CBCC2 on f_4

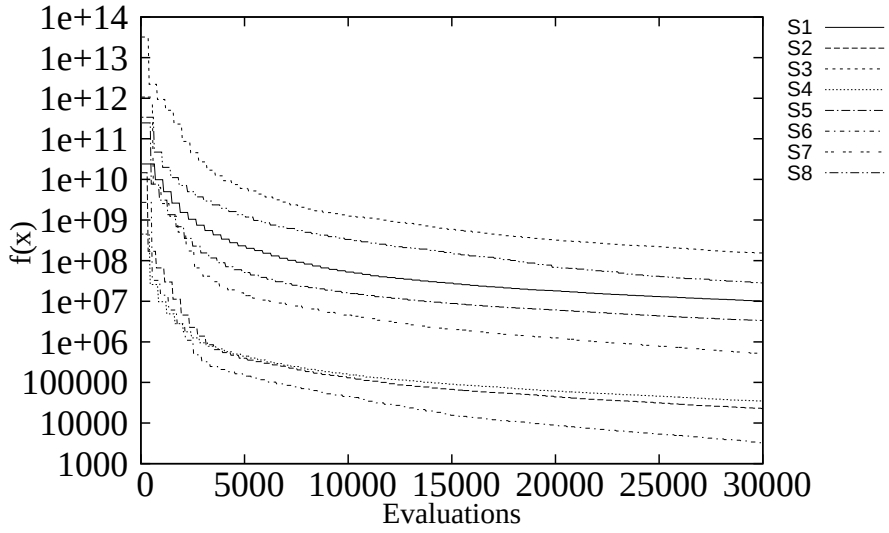
(c) CBCC1 on f_5 (d) CBCC2 on f_5

Figure 84: Number of evaluations used to optimize components for CBCC1 and CBCC2 on f_4 and f_5 (CEC'13 LSGO)

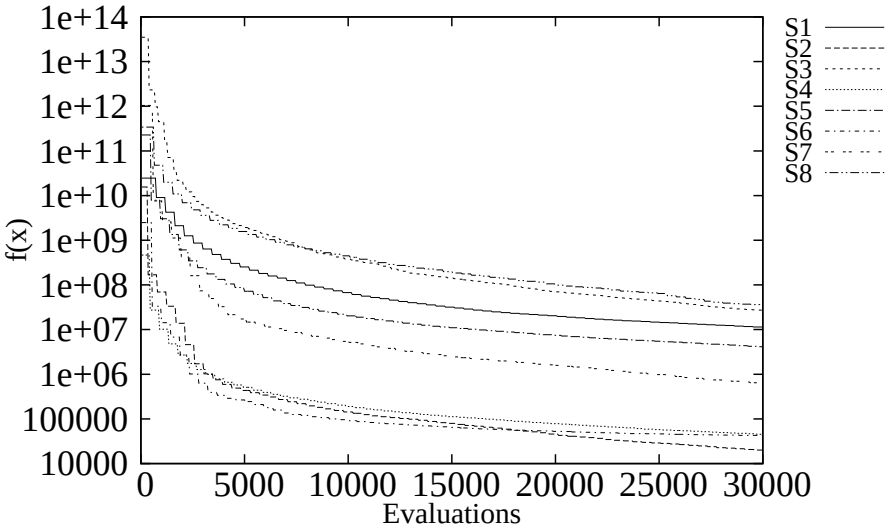
than both CBCC1 and round-robin CC. The main reason for this type of unexpected result can be unleashed by inspecting the convergence plots of individual components which are illustrated in Figures 85 and 86 for f_4 and f_8, respectively.

As Figures 85 and 86 demonstrate, the component with the largest weight has an initial large objective value. Therefore, CBCCs optimize this component more frequently than the remaining subproblems. This causes a rapid drop in the objective value of this particular component which is visible in Figures 85c and 86c. When the objective value of the chosen component drops below those of other components, CBCC should stop its optimization in order to give a chance to other subproblems which have relatively a higher potential for further improvements.

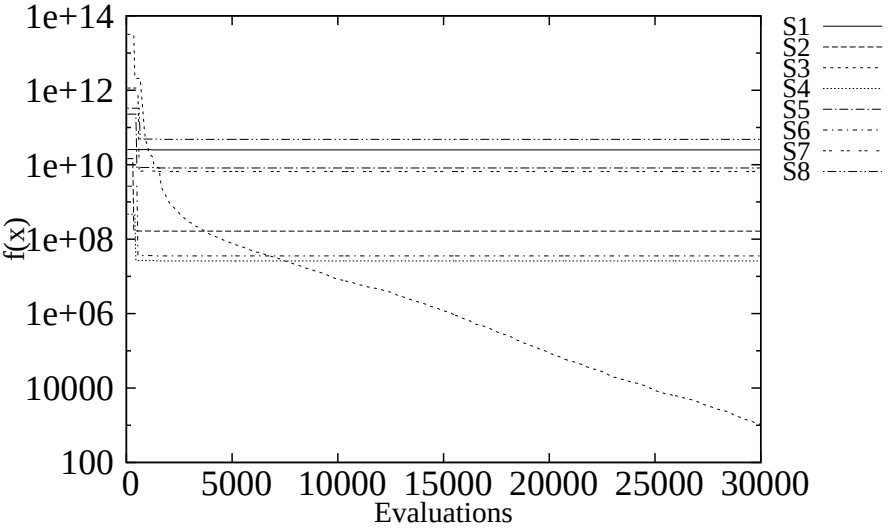
As we observe in Figures 85c and 86c, CBCC2 fails to detect this important event which we call *the crossing moment* since the convergence curve of the selected subproblem crosses one or more other curves. Failing to promptly responding to this event is the reason behind CBCC2's poor performance. The root cause of such behavior can be found in Algorithm 2. As the pseudocode shows, the termination criteria of CBCC2's exploitation phase is nothing but the complete stagnation of the selected subpopulation ($\min(\mathbf{f}_c) \not< \min(\mathbf{f}_p)$). In other words, as long as the chosen subproblem continues improving, even very marginally, it will be optimized further by CBCC2. This means even if the immediate improvement gained from optimizing the component that used to be the most contributing subfunction is less than all other components but still positive, CBCC2 will not switch to any other subfunction.



(a) CC on f_4

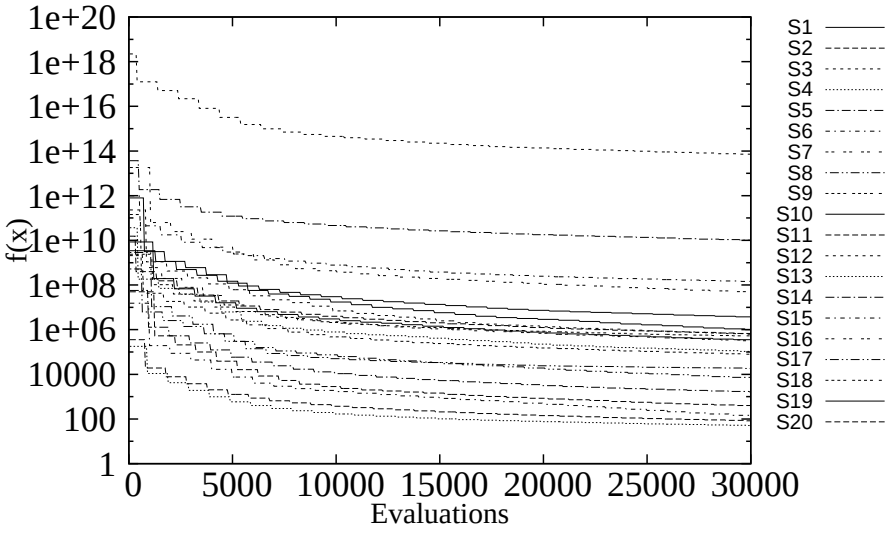


(b) CBCC1 on f_4

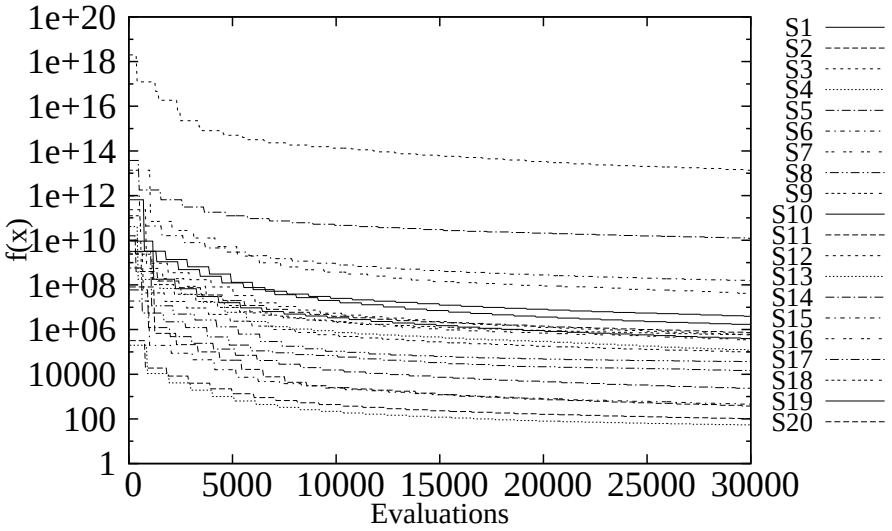


(c) CBCC2 on f_4

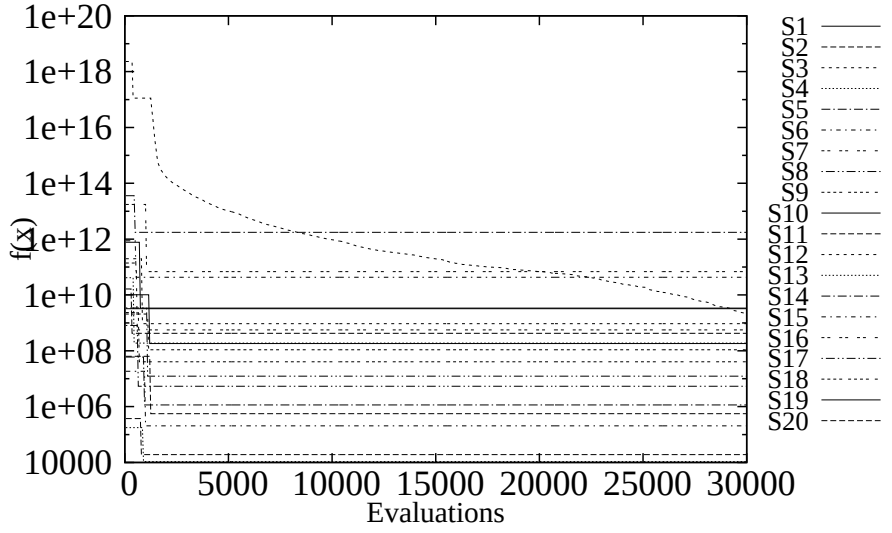
Figure 85: Convergence plots of individual components of f_4 for CC, CBCC1 and CBCC2 (CEC'13 LSGO).



(a) CC on f_8



(b) CBCC1 on f_8



(c) CBCC2 on f_8

Figure 86: Convergence plots of individual components of f_8 for CC, CBCC1 and CBCC2 (CEC'13 LSGO).

Another problem that is associated with CBCC2 is that even if we force it to break out of the inner loop and switch to exploration phase, there is still a fair chance that the same subfunction to be selected for the next exploitation phase, due to the accumulation of its contributions from the very beginning rounds of optimization (see $\boldsymbol{\mu}[k] \leftarrow \boldsymbol{\mu}[k] + \min \mathbf{f}_p - \min \mathbf{f}_c$ in Algorithm 2). This results in deprivation of other components that have higher contributions in the intermediary stages of optimization. It should be noted CBCC1 is also prone to this shortcoming, but it takes more time to become evident on the convergence plots at it switches to exploration phase very frequently.

In general, two major drawbacks of CBCCs can be summarized as follows:

1. The CBCCs' slow response to local changes in the fitness value and their strong reliance on the information accumulated from the early stages of the optimization process. This is more evident in CBCC2 than CBCC1.

2. Over-exploration by frequent application of the exploration phase in Algorithm 2. This is an inefficient use of the limited resources. This behavior is more evident in CBCC1 and it is the main reason behind the similar performance of CBCCs and CC in some cases.

In the next section, we explain how these issues can be addressed by only considering the most recent feedback (instead of accumulated rewards) and introducing a simple probabilistic exploration mechanism. We cite this new variant of CBCC framework as CBCC3.

8.3.2 CBCC3: Improving Exploration-Exploitation Balance

In the previous section, we identified two major limitations of CBCCs. This section proposes a new version of CBCC that addresses these shortcomings. Algorithm 6 presents the details of the proposed algorithm (*i.e.,* CBCC3) which to a large extent resembles the previous CBCC variants (see Algorithm 2).

Algorithm 6 shows that there are three major differences between CBCC3 and its predecessors:

1. CBCC3 does not run the exploration phase in every coevolutionary cycle. As line 7 of the algorithm confirms, CBCC3 enters to the exploration phase with a probability of p_t (note that at least one execution of this phase is guaranteed at the first cycle). Based on our observations in the previous section, the exploration phase should happen with a relatively low probability to prevent waste of computational budget as we observe in CBCC1. The analysis of the sensitivity of CBCC3 performance to p_t will be postponed to Section 8.3.3.

2. CBCC3 only considers the most recent improvements as the estimated contributions and eliminates the use of historical information as opposed to CBCC1 and CBCC2. As mentioned at lines 12-13 and 19-20 of the algorithm, the last non-zero difference in the objective values of two consecutive iterations is recorded as the contribution of a given subfunction (*i.e.,* $\dot{\boldsymbol{\delta}}_k$). CBCC3 will use these values to select the most contributing component for further optimization in the upcoming exploitation phases (line 14).

3. The exploitation phase of CBCC3 continues until the current most contributing component loses its position. In other words, if the latest improvement gained by optimizing subfunction f_k^* is less than the improvement previously recorded for

Algorithm 6 Contribution-Based Cooperative Coevolution 3

function CBCC3(f, D, N, p_t)
  $(\mathbf{X}, \mathbf{f}_p) \leftarrow$ `initialization`(f, N, D) ▷ Initialization
  $(\mathcal{D}, K) \leftarrow$ `initGrouping`(f) ▷ Decomposition
  $\vec{\mathbf{x}} \leftarrow$ `initContextVector`$(\mathbf{X}, \mathbf{f}_p)$ ▷ Initial *Context*
  $\dot{\boldsymbol{\delta}} \leftarrow$ `zeros`$(1, K)$ ▷ Initial contributions
  while `termination`$() \neq$ `True` **do** ▷ **Main loop**
    if $\sum_{k=1}^{K} \dot{\boldsymbol{\delta}}[k] = 0$ **or** `rand`$() < p_t$ **then** ▷ **Exploration Phase**
      for $k \leftarrow 1 \cdots K$ **do**
        $\mathbf{f}_p \leftarrow \mathbf{f}_c$
        $(\mathbf{X}, \mathbf{f}_c, \vec{\mathbf{x}}) \leftarrow$ `optimize`$(f, \mathbf{X}, \vec{\mathbf{x}}, \mathcal{D}, k, \mathbf{f}_p)$ ▷ 1 epoch optimization
        $\dot{\delta} \leftarrow \min(\mathbf{f}_p) - \min(\mathbf{f}_c)$ ▷ Latest improvement
        if $\dot{\delta} > 0$ **then**
          $\dot{\boldsymbol{\delta}}[k] \leftarrow \dot{\delta}$ ▷ Update contribution
    $\overset{*}{k} \leftarrow \arg\max(\dot{\boldsymbol{\delta}})$ ▷ Select a component
    while $\overset{*}{k} = \arg\max(\dot{\boldsymbol{\delta}})$ **do** ▷ **Exploitation Phase**
      $\mathbf{f}_p \leftarrow \mathbf{f}_c$
      $(\mathbf{X}, \mathbf{f}_c, \vec{\mathbf{x}}) \leftarrow$ `optimize`$(f, \mathbf{X}, \vec{\mathbf{x}}, \mathcal{D}, \overset{*}{k}, \mathbf{f}_p)$ ▷ 1 epoch optimization
      $\dot{\delta} \leftarrow \min(\mathbf{f}_p) - \min(\mathbf{f}_c)$ ▷ Latest improvement
      if $\dot{\delta} > 0$ **then**
        $\dot{\boldsymbol{\delta}}[\overset{*}{k}] \leftarrow \dot{\delta}$ ▷ Update contribution
      if `termination`$() =$ `True` **then**
        break
  return $\mathbf{X}[\arg\min(\mathbf{f}_c), :]$ ▷ Return final solution

optimizing any other component, CBCC3 stops the exploitation phase and switches to exploration phase. Recall that CBCC1 executes the exploitation phase once and CBCC2 repeatedly runs it until $f_{\overset{*}{k}}$ stops improving. This special consideration in termination of exploitation phase in CBCC3 and entering next exploration phase ensures that the crossing moment that was observed in CBCC2's convergence plots (see Figures 85c and 86c) does not happen in CBCC3. Therefore, we can expect that CBCC3 executes the exploitation phase more often than CBCC1 but not as long as in CBCC2's exploitation phase.

8.3.3 Experiments and Analysis

In this section, we compare the performance of CBCC3 with round-robin CC, CBCC1 and CBCC2 using a subset of the CEC'13 LSGO benchmark set. The total number of fitness evaluations is set to 3×10^6 as suggested in [Li et al. 2013a]. As usual, we eploy SaNSDE [Yang et al. 2008c] as the component optimizer while we fix the epoch length to 100 iterations, and the population size N to 50.

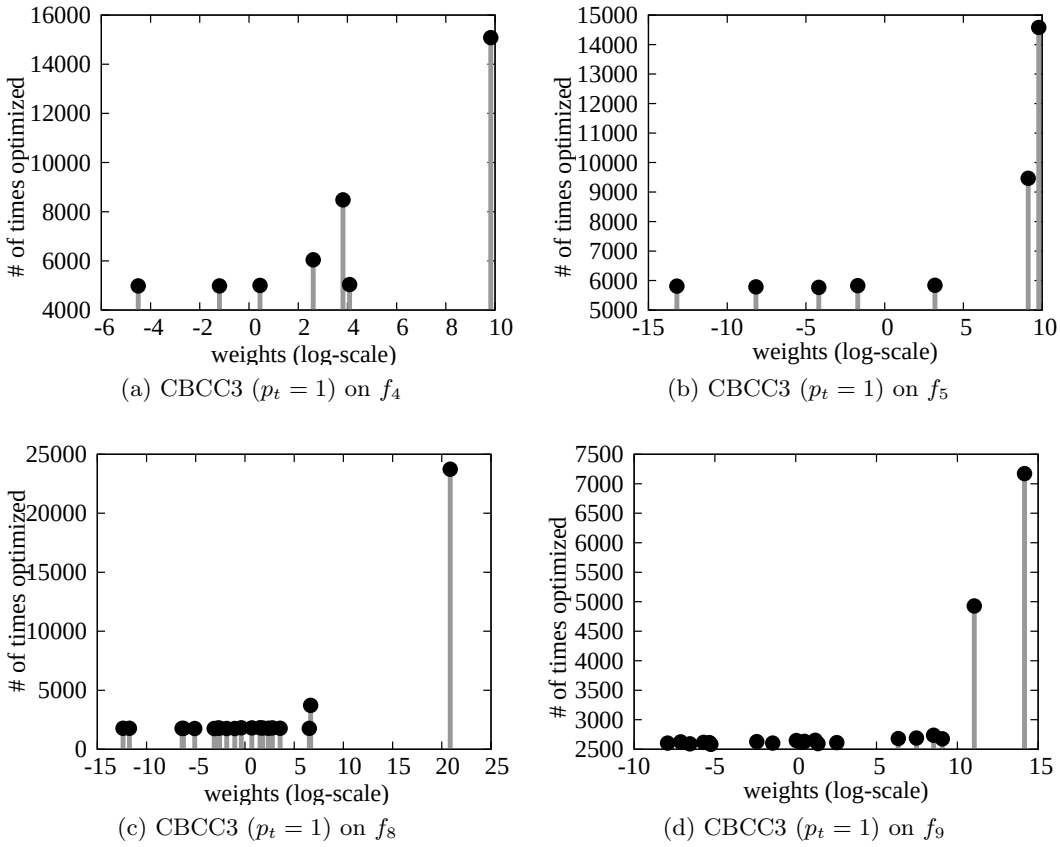
Table 87 presents the median, mean and standard deviation of 25 independent runs for the aforementioned algorithms. For the statistical significance of the differences between the algorithms' performance, we first use Kruskal-Wallis [Sheskin 2003] with a 95% confidence interval. In the cases that Kruskal-Wallis test detects a significant difference among the given algorithms, we run a series of pair-wise Wilcoxon rank-sum tests and apply Holm p-value correction to account for the family-wise error rate [Derrac et al. 2011]. The **bold** entries in Table 87 denote the cases that the CBCC3 performs significantly

Table 87: Comparison between CBCC3 and CBCC1, CBCC2, and round-robin CC on f_4-f_{11} (CEC'13 LSGO). The **bold** entries are significantly better based on pair-wise Wilcoxon rank-sum with Holm p-value correction ($\alpha = 0.05$). m, μ, and σ denote median, mean, and standard deviations of independent runs.

					CBCC3		
		CC	CBCC1	CBCC2	$p_t = 1$	$p_t = 0$	$p_t = 0.05$
f_4	m	1.53e+08	6.54e+07	9.03e+10	3.61e+07	**2.18e+07**	**2.51e+07**
	μ	1.97e+08	7.71e+07	8.77e+10	4.08e+07	2.20e+07	2.97e+07
	σ	1.51e+08	4.05e+07	1.14e+10	2.09e+07	8.05e+06	1.56e+07
f_5	m	2.65e+06	2.29e+06	**2.06e+06**	2.23e+06	**2.16e+06**	**1.97e+06**
	μ	2.66e+06	2.28e+06	2.09e+06	2.34e+06	2.13e+06	1.99e+06
	σ	7.12e+05	3.55e+05	3.52e+05	4.70e+05	3.49e+05	3.61e+05
f_6	m	8.74e+04	8.74e+04	8.35e+04	8.74e+04	8.74e+04	8.35e+04
	μ	8.57e+04	8.85e+04	8.39e+04	8.65e+04	8.45e+04	7.94e+04
	σ	1.95e+04	2.88e+04	2.36e+04	1.88e+04	1.92e+04	3.43e+04
f_7	m	4.53e+07	6.23e+07	7.85e+07	4.68e+07	**3.86e+05**	**3.27e+05**
	μ	5.12e+07	6.38e+07	8.82e+07	4.75e+07	2.09e+07	1.42e+07
	σ	3.67e+07	4.01e+07	6.78e+07	3.38e+07	3.04e+07	2.18e+07
f_8	m	5.43e+13	1.09e+13	1.90e+12	7.29e+10	**2.28e+09**	**5.47e+09**
	μ	7.19e+13	1.38e+13	1.88e+12	1.51e+11	1.21e+10	8.23e+09
	σ	6.07e+13	1.14e+13	2.80e+11	2.87e+11	2.40e+10	1.03e+10
f_9	m	2.95e+08	2.34e+08	2.00e+08	2.08e+08	**1.42e+08**	**1.58e+08**
	μ	2.85e+08	2.32e+08	2.03e+08	2.02e+08	1.40e+08	1.56e+08
	σ	6.20e+07	4.85e+07	2.45e+07	5.09e+07	1.55e+07	3.51e+07
f_{10}	m	7.05e+01	7.51e+01	7.17e+01	7.68e+01	7.46e+01	7.30e+01
	μ	6.90e+01	7.44e+01	7.16e+01	7.66e+01	7.44e+01	7.15e+01
	σ	1.68e+01	9.97e+00	1.36e+01	1.24e+01	1.07e+01	1.49e+01
f_{11}	m	1.51e+10	1.41e+09	1.44e+09	1.07e+09	**4.49e+08**	**6.31e+08**
	μ	2.62e+10	1.58e+10	1.63e+10	1.33e+09	4.74e+08	6.24e+08
	σ	3.10e+10	2.26e+10	2.76e+10	1.41e+09	2.95e+08	3.47e+08

better than others. In order to investigate the sensitivity of CBCC3 to its key parameter p_t, we run CBCC3 with three different p_t values: $p_t \in \{0, 0.05, 1\}$. Note that $p_t = 0$ switches off the exploration phase during the optimization. Therefore, this phase is only executed once at the beginning of the algorithm. On the other hand, $p_t = 1$ makes the CBCC3's exploration phase very similar to other variants of CBCC as it ensures that every exploitation phase will be followed by an exploration phase. Table 87 strongly suggests that CBCC3 is superior to its predecessors as well as the standard round-robin CC.

For further investigations on CBCC3's behavior, we present a series of plots in Figures 87–89 to demonstrate how the computational resources are allocated to various components with respect to their coefficients (similar to Figure 84). As shown in the figures, the allocation pattern for $p_t = 1$, which causes the exploration phase to occur at every cycle, is very similar to that of CBCC1. For a clearer comparison between CBCC3 with different p_t values and other algorithms, we provide the Win-Tie-Lose scores in Table 88



(a) CBCC3 ($p_t = 1$) on f_4

(b) CBCC3 ($p_t = 1$) on f_5

(c) CBCC3 ($p_t = 1$) on f_8

(d) CBCC3 ($p_t = 1$) on f_9

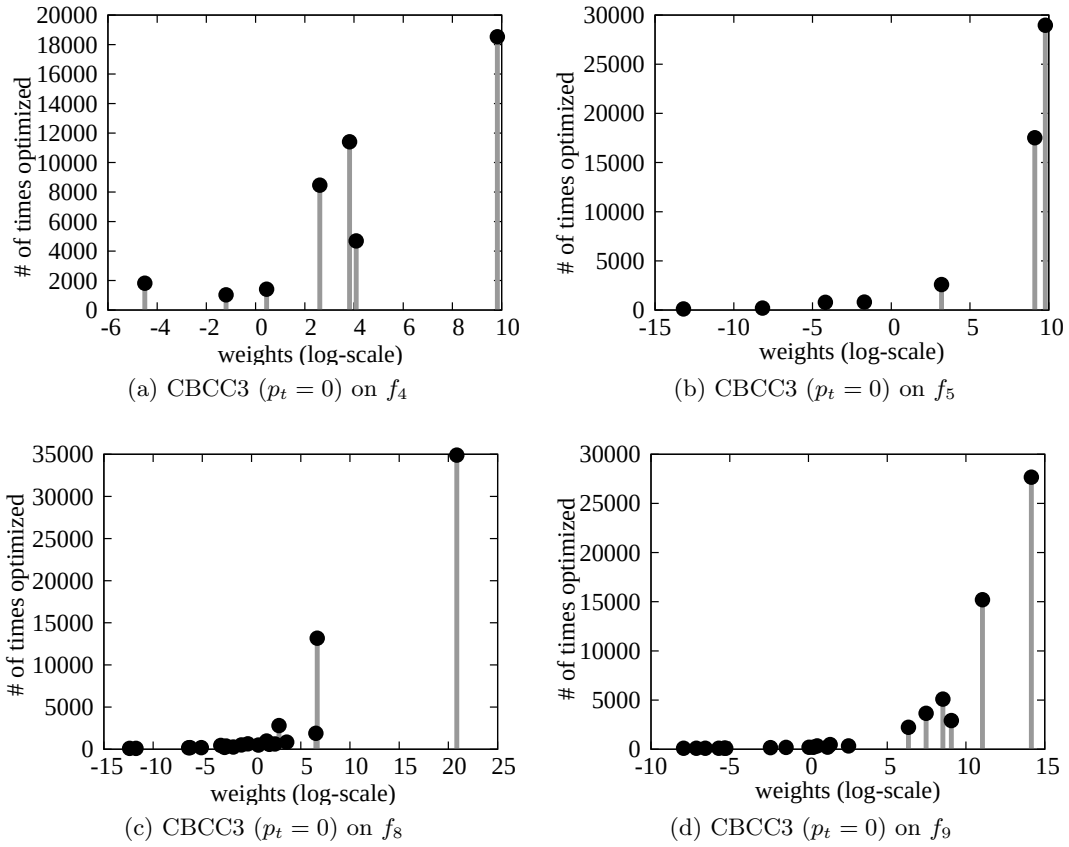
Figure 87: Number of evaluations CBCC3 with $p_t = 1$ used to optimize each component of f_4, f_5, f_8, and f_9 (CEC'13 LSGO).

using a pair-wise Wilcoxon rank-sum with 95% confidence interval.

Table 88: Number of wins, ties, and loses (W-T-L) between all pairs of algorithms using Wilcoxon rank-sum ($\alpha = 0.05$).

					CBCC3	
	CC	**CBCC1**	**CBCC2**	$p_t = 0$	$p_t = 0.05$	$p_t = 1$
CC		4-4-0	3-3-2	5-3-0	6-2-0	4-4-0
CBCC1	0-4-4		3-4-1	5-3-0	6-2-0	4-4-0
CBCC2	2-3-3	1-4-3		5-3-0	5-3-0	4-3-1
$p_t = 0$	0-3-5	0-3-5	0-3-5		0-8-0	0-3-5
CBCC3 $p_t = 0.05$	0-2-6	0-2-6	0-3-5	0-8-0		0-2-6
$p_t = 1$	0-4-4	0-4-4	1-3-4	5-3-0	6-2-0	

As Table 88 confirms, all variants of CBCC3 outperform CC, CBCC1 and CBCC2 in most cases. In the case of $p_t = 1$, CBCC3 has more ties with the other algorithms. This is consistent with what we observed from Figures 87–89. In this case, the difference between CBCC3 and other algorithms comes from the way the contributions are quantified as well as the termination criteria of the exploitation phase. Table 88 clearly shows that the new budget allocation strategy improves the overall efficiency of CBCC. We also observe that the performance of CBCC3 with $p_t = 0$ is better than CBCC3 with $p_t = 1$ which confirms



(a) CBCC3 ($p_t = 0$) on f_4

(b) CBCC3 ($p_t = 0$) on f_5

(c) CBCC3 ($p_t = 0$) on f_8

(d) CBCC3 ($p_t = 0$) on f_9

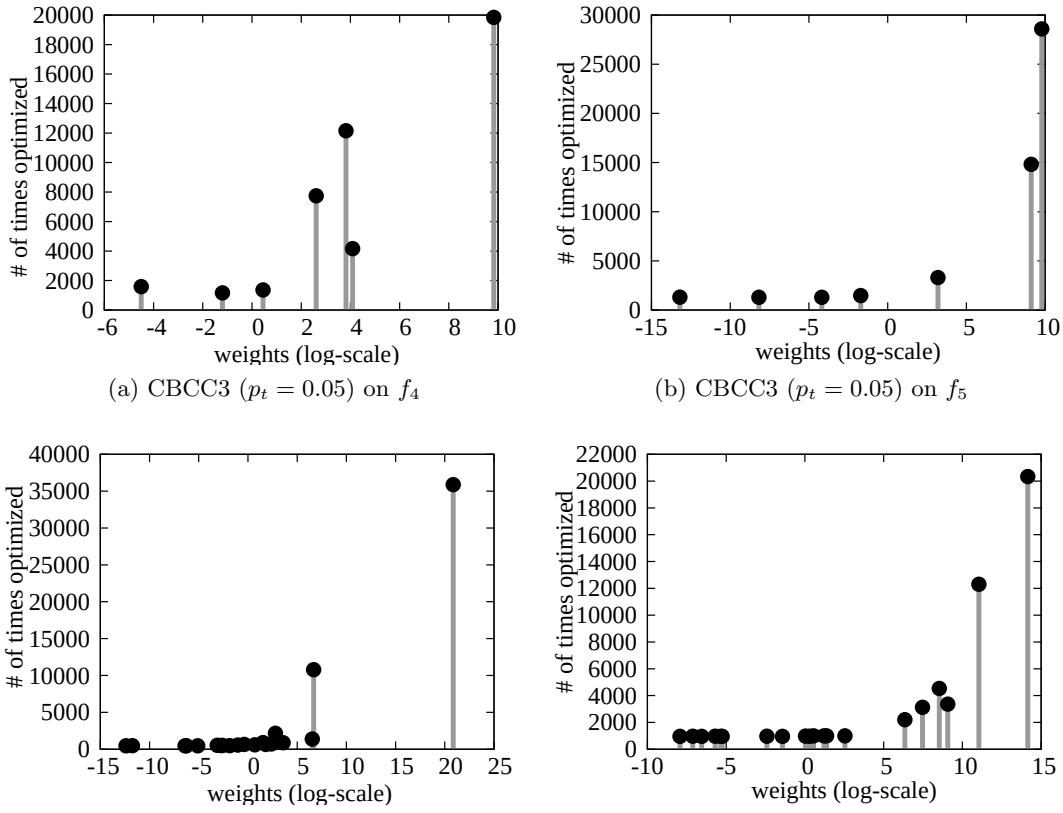
Figure 88: Number of evaluations CBCC3 with $p_t = 0$ used to optimize each component of f_4, f_5, f_8, and f_9 (CEC'13 LSGO).

that over-exploration can adversely affect the performance of the CBCC (*i.e.,* especially in the case of CBCC1 and CBCC3 with $p_t = 1$). However, we can see that very occasional explorations ($p_t = 0.05$) is beneficial.

Figures 810–811 present the convergence plots of individual components of CBCC3 using various p_t values on f_4 and f_6. We can see that the crossing moment that happen in CBCC2 (see Figure 85c) does not occur for CBCC3. We also observe that the convergence plots are more concentrated especially in the f_6 case. To quantitatively point out this difference between CBCC variantss, we reported the standard deviation among the final objective values of all components in Table 89.

It is expected that an efficient budget management schema minimizes the variation between the objective value of individual components. With respect to the convergence plots (*e.g.,* Figures 810–811), we expect that an effective budget allocation schema to force the objectives value of all subfunctions to converge to similar values. In other words, if the objective value of one component drops below the objective values of other components, we can conclude that its contribution to the overall objective value becomes negligible (assuming that the other components are not stagnant). Therefore, an effective budget allocation schema should switch to other components to force the objective values of all subfunctions become and remain close to each other. If this does not happen, it means that the resource allocator is not successful in selecting the best component for optimization (except in the cases that a sudden drop in objective value occurs at the very late cycle).

Table 89 demonstrates the strong correlation between the performance of CBCC and



(a) CBCC3 ($p_t = 0.05$) on f_4

(b) CBCC3 ($p_t = 0.05$) on f_5

(c) CBCC3 ($p_t = 0.05$) on f_8

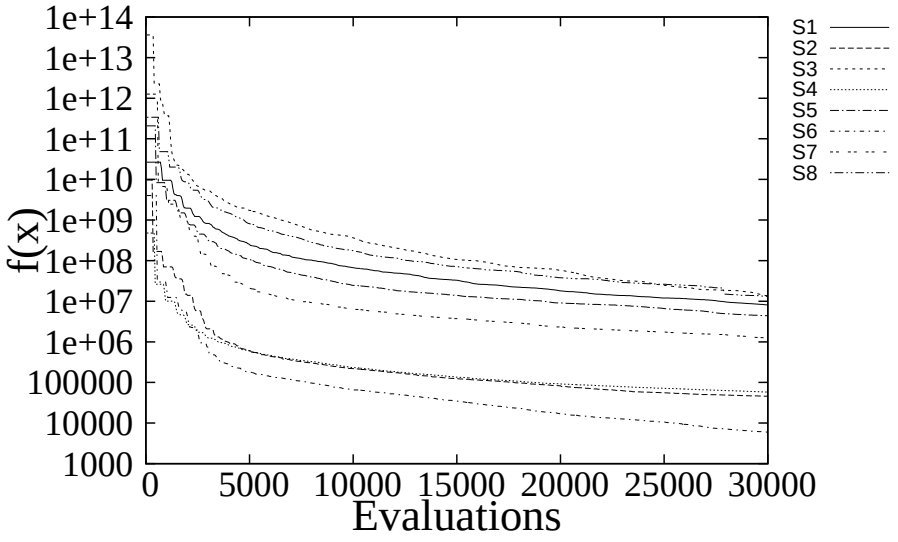
(d) CBCC3 ($p_t = 0.05$) on f_9

Figure 89: Number of evaluations CBCC3 with $p_t = 0.05$ used to optimize each component of f_4, f_5, f_8, and f_9 (CEC'13 LSGO).

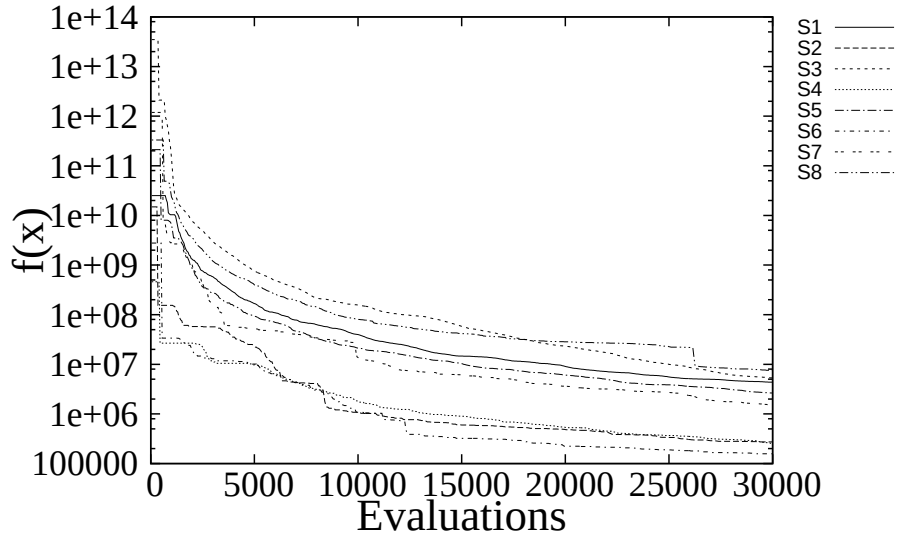
the variability among the subfunctions' objective values. For the sake of simplicity, the medians of 25 independent trials from Table 87 are repeated in Table 89. In this table, the entry for the algorithm with the smallest median in overall objective value is made **bold**. The bottom half of this table reports the standard deviation among the final objective values of individual components across all the 25 independent trials. The **bold** numbers indicate entries with the smallest standard deviation. It should be noted that these are different from the standard deviations of the 25 independent runs reported in Table 87.

In Table 89, we observe a strong correlation between low variability among the objective values of individual subfunctions and the overall objective value except for f_6, f_8, and f_{10}. With respect to Table 87 we know that all algorithms perform similarly on functions f_6 and f_{10}. Therefore, the only exception is problem f_8. We can conclude that CBCC3 with a small positive p_t value results in a lower variability among its components. This has a direct relationship with the concentration of convergence plots in Figures 810–811 and its overall good performance as shown in Table 87.

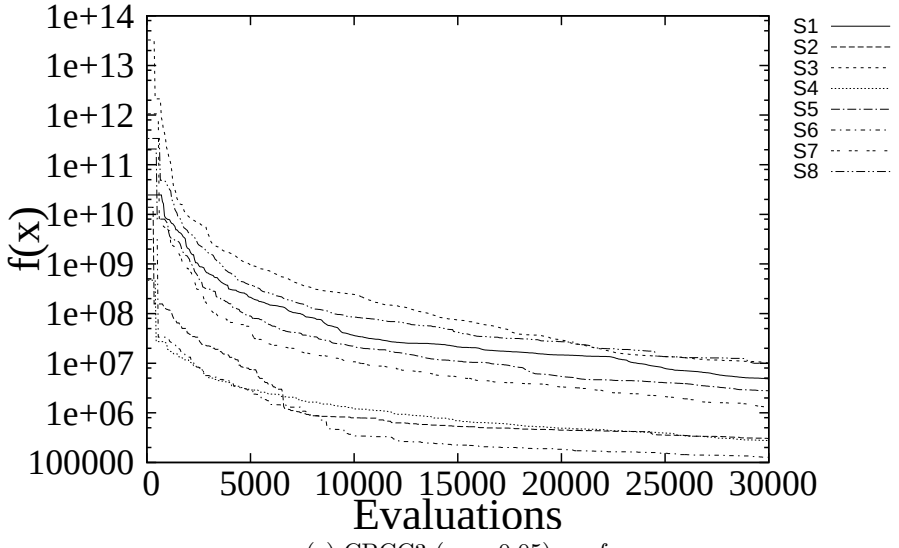
Although CBCC3 demonstrates significant improvement over its predecessors and the round-robin CC, its statistically similar performance on f_6 and f_{10} requires further investigations. The convergence plots of CBCC3 on f_6 suggests that the objective value of several subfunctions have an initial improvement and stays unchanged throughout the optimization process. Since the magnitude of the objective value of these components is relatively larger than other components, we expect that CBCC3 allocates a considerable portion of the available computational resources to these components. From Algorithm 6



(a) CBCC3 ($p_t = 1$) on f_4

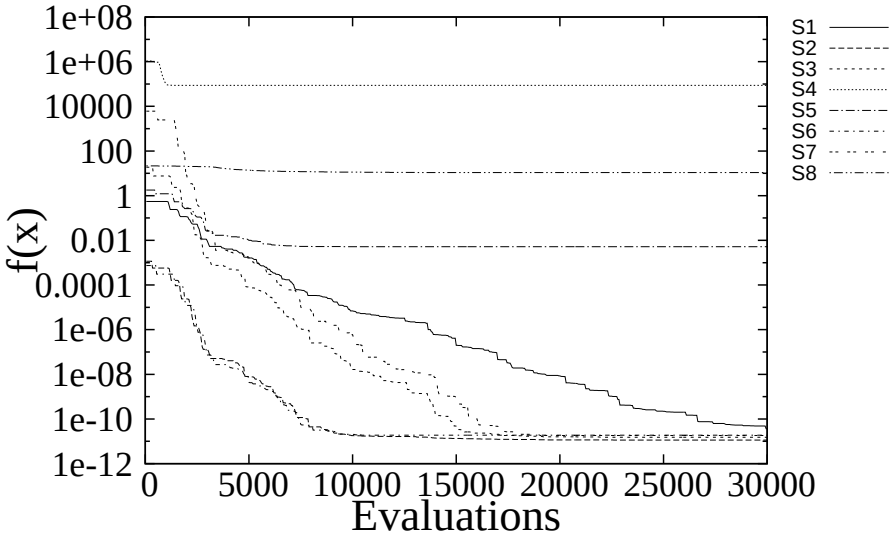


(b) CBCC3 ($p_t = 0$) on f_4

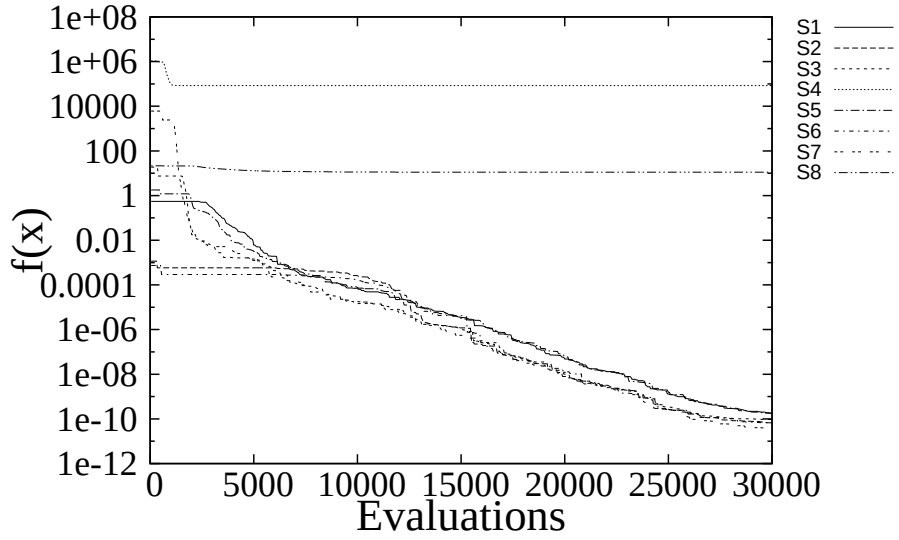


(c) CBCC3 ($p_t = 0.05$) on f_4

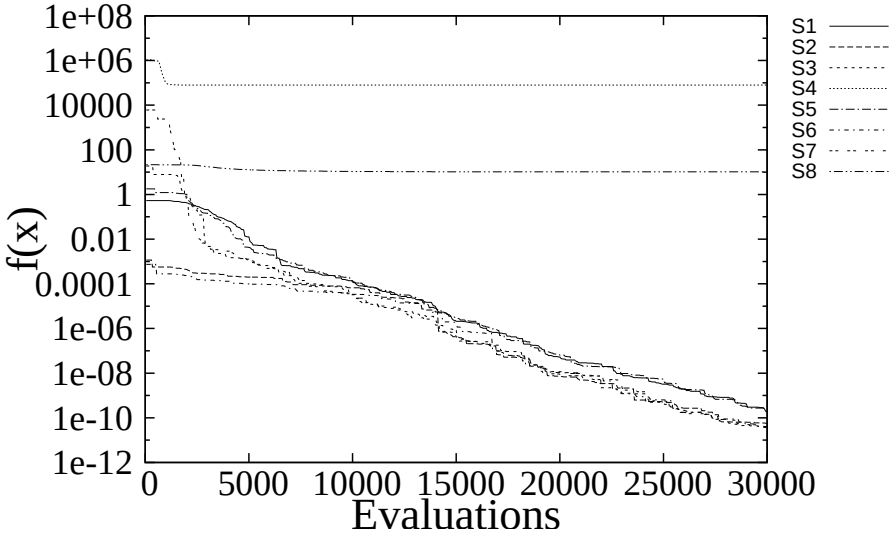
Figure 810: Convergence plots of individual components of f_6 for CBCC3 with $p_t \in \{0, 0.05, 1\}$ (CEC'13 LSGO).



(a) CBCC3 ($p_t = 1$) on f_6



(b) CBCC3 ($p_t = 0$) on f_6



(c) CBCC3 ($p_t = 0.05$) on f_6

Figure 811: Convergence plots of individual components of f_6 for CBCC3 with $p_t \in \{0, 0.05, 1\}$ (CEC'13 LSGO).

Table 89: Volatility in the final objective value of individual components. CBCC3 maintains a lower STD than the others. There is also a strong correlation between better overall performance and lower STD.

	Median					
				CBCC3		
	CC	**CBCC1**	**CBCC2**	$p_t = 1$	$p_t = 0$	$p_t = 0.05$
f_4	1.53e+08	6.54e+07	9.03e+10	3.61e+07	**2.18e+07**	2.51e+07
f_5	2.65e+06	2.29e+06	2.06e+06	2.23e+06	2.16e+06	**1.97e+06**
f_6	8.74e+04	8.74e+04	8.35e+04	8.74e+04	8.74e+04	**8.35e+04**
f_7	4.53e+07	6.23e+07	7.85e+07	4.68e+07	3.86e+05	**3.27e+05**
f_8	5.43e+13	1.09e+13	1.90e+12	7.29e+10	**2.28e+09**	5.47e+09
f_9	2.95e+08	2.34e+08	2.00e+08	2.08e+08	**1.42e+08**	1.58e+08
f_{10}	7.05e+01	7.51e+01	7.17e+01	7.68e+01	7.46e+01	**7.30e+01**
f_{11}	1.51e+10	1.41e+09	1.44e+09	1.07e+09	**4.49e+08**	6.31e+08
	Standard Deviation					
				CBCC3		
	CC	**CBCC1**	**CBCC2**	$p_t = 1$	$p_t = 0$	$p_t = 0.05$
f_4	7.28e+07	1.75e+07	1.63e+10	8.29e+06	**3.68e+06**	5.95e+06
f_5	8.04e+05	6.56e+05	5.90e+05	6.94e+05	6.19e+05	**5.75e+05**
f_6	2.92e+04	3.10e+04	2.90e+04	2.94e+04	**2.88e+04**	2.89e+04
f_7	1.74e+07	1.91e+07	2.54e+07	1.65e+07	1.09e+07	**7.54e+06**
f_8	2.06e+13	3.91e+12	3.87e+11	7.01e+10	5.52e+09	**2.47e+09**
f_9	5.53e+07	4.31e+07	3.60e+07	3.83e+07	**2.51e+07**	2.95e+07
f_{10}	1.04e+01	1.09e+01	1.09e+01	1.13e+01	**1.09e+01**	**1.09e+01**
f_{11}	8.72e+09	5.89e+09	6.86e+09	3.62e+08	**9.31e+07**	1.16e+08

we know that the exploration phase is executed with a probability (*i.e.*, p_t) only if the algorithm breaks out of the exploitation phase. The convergence plots do not show whether this occurs because of natural stagnation of these components or because of the lack of exploration due to dominance of particular components.

A drawback of CBCC3 is that it only relies on the magnitude of components' contributions for its selection policy and does not approximate the likelihood of their realization. To address this limitation, one can borrow some ideas from online resource allocation techniques that are common in reinforcement learning area. We explore this direction of research in Chapter 10.

8.4 Chapter Summary

We started this chapter by studying the sensitivity of CBCC1 and CBCC2 in terms of two significantly influential factors: the decomposition accuracy and the imbalance level. Our investigations revealed that CBCC1 is less sensitive to these factors than CBCC2 as it improved conventional round-robin CC in most cases. As a result, we recommend using CBCC1 instead of traditional non-contribution-aware techniques even when the

decomposition error is inevitable and the imbalance level is unknown or minor.

We performed a number of case studies and showed that CBCC1 suffers from over-exploration by excessive application of the exploration phase in which the contribution of all components is recalculated. Contrary to CBCC1, CBCC2 suffers from over-exploitation by greedily continuing the optimization of the component with the highest estimated contribution until it becomes stagnant. Our experimental results showed that the accumulation of contributions from the first cycle which happens in CBCC1 and CBCC2 biases their selection mechanism towards the subfunction with an initial good contribution. To address these issues, we proposed the CBCC3 algorithm which maintains be better exploration-exploitation balance in the component space. Our numerical simulations confirm that CBCC3 performs significantly better on large-scale imbalanced functions than the round-robin CC and older variants of CBCC.

In CBCC3 a component is optimized until its immediate contribution drops below the last recorded contribution of another component. In addition, CBCC3 completely eliminates the accumulation of contributions and selects a component only based on its most recent recorded improvement. Furthermore, the exploration phase in CBCC3 occurs with a small probability to avoid the waste of limited resources on recalculation of contributions.

There are several lines of research arising from this work which worth pursuing in future. Firstly, one should study the sensitivity of broader family of CACC to a wide range of decomposition errors. It is expected that different types errors show different effects on the performance of budget allocation technique. We reviewed some of the possible grouping errors in Subsection 8.2.1, but the space limitation prevents us from further expansions. Another obstacle to conducting such massive analyses in that the current benchmarks (*e.g.,* CEC'13 LSGO) are very limited in several ways. In Chapter 9 we discuss the limitations of current test sets and propose the most comprehensive set of benchmark problems that covers different cases of imbalanced functions.

Secondly, one can investigate the sensitivity of CC (in general) and CACCs (in particular) to the epoch length. Although all CC algorithms have this parameter in common, no empirical studies have been published on the effect of epoch length when a CC or CACC is employed to solve imbalanced functions. Our preliminary experimental results show that as the epoch length grows the accuracy of contribution estimation increases since the uncertainty in the contribution estimation decreases. However, long epochs may increase the chance of concept drift as the estimations are updated less frequently and thus the algorithms respond to the changes very slowly. In Chapter 10, we present some brief sensitivity analyses on the effect of epoch length on the overall performance of some CC and CACC variants.

Finally, the CBCC framework can be improved in a number of ways. Although CBCCs, and especially CBCC3, have shown very promising performance in dealing with imbalanced functions, they still suffer from some limitations. One of the shortcomings of this framework is that all of its variations only rely on the magnitude of components' contributions and do not approximate the likelihood of their realization. Additionally, the *ad hoc* heuristics that used in CBCCs may perform well but are hard to be justified, studied, and expanded. We certainly need a more general framework that brings some flexibility to the algorithms. In Chapter 10, we reformulate the resource allocation problem in CC framework as a dynamic multi-armed bandit problem. Then, we propose a flexible modular framework that can solve this problem in the most efficient way.

CHAPTER 9

Designing Benchmark Problems with Contribution Imbalance

In this chapter, we review the previously proposed benchmark functions for box-constrained large-scale continuous problems and discuss the means they provide to study imbalanced components. Then, we investigate the limitations of the available test sets by pointing out the imbalance scenarios that they do not cover. To remedy these shortcomings, we propose a set of 40 large-scale problems which form eight groups each of which contains five functions. Then, we explain the features of each category in detail. Finally, we perform several numerical experiments using this new benchmark set to demonstrate the facilities it offers for further studies on partially separable large-scale functions with imbalanced components.

9.1 Introduction

During the past decade, a number of test sets have been proposed to promote and standardize the researches on large-scale optimization problems. These suites have been widely used in a variety of studies. The first set was the Benchmark Functions for the IEEE CEC 2008 Special Session and Competition (*a.k.a.* CEC'08) [Tang et al. 2007]. The suite contains seven scalable functions from a combination of fully separable and nonseparable functions. The major limitations of CEC'08 are the lack of modularity, inadequate diversity, and no inclusion of functions with imbalanced components [Omidvar 2015]. Therefore, despite its scalability and popularity, CEC'08 is not particularly useful for investigations on the issue of contribution imbalance.

Two years later, the Benchmark Functions for the IEEE CEC 2010 Special Session and Competition (*a.k.a.* CEC'10) was proposed to address the shortcomings of CEC'08. The set consists of 20 high-dimensional problems from a mixture of functions with different degrees of separability [Tang et al. 2009]. Among the 15 partially separable functions in this set, 10 of them reflect the contribution imbalanced problem in terms of unequal subfunction sizes. Although there are three imbalanced categories, all of them follow a very similar pattern. In all cases, a few (*i.e.,* either 1 or 10) nonseparable components with 50 dimensions are combined with one fully separable component of size 500 or 950. The set also contains a category of uniform partially separable problems with basis functions similar to imbalanced functions. Nevertheless, the set did not include any problem with other types of contribution imbalance such as components with unequal coefficients. Fur-

thermore, the size of all partially separable components is fixed to 50. One can conclude that CEC'10 poorly represent real-world scenarios as it is very unlikely to have problems with the same component sizes, coefficients, and subfunction features.

Finally, the Benchmark Functions for the IEEE CEC 2013 Special Session and Competition (*a.k.a.* CEC'13) was proposed that includes 15 large-scale problems [Li et al. 2013a]. The optimum value of all problem instances in this set are shifted to arbitrary locations, their symmetry (if existed) is broken and some irregularities are added to the search landscapes. The CEC'13 contains 10 partially separable functions which all of them are imbalanced in terms of unequal dimension sizes and nonuniform coefficients. This set is the current research standard for studies on large-scale optimization.

A few studies on imbalance issue have used CEC'13 in its original format [Mahdavi et al. 2016c]. Some other researchers, however, modified CEC'13 as it does not cover all imbalanced sources and scenarios. For example, there is no function with heterogeneous subfunctions, the subproblem sizes and coefficients were chosen arbitrarily, and there was no balanced counterpart to compare contribution-aware (CACC) algorithms in both balanced and imbalanced scenarios. These shortcomings motivated us to expand CEC'13 to a more comprehensive test set particularly for further studies on the topic of imbalanced contributions.

The key features that we believe a comprehensive benchmark set on imbalance components should have are as follows:

- **Diverse basis functions:** Since each optimizer may perform better on some subclasses of problems, a fair benchmark set should include a variety of basis functions to decrease the chance of biased conclusions. In fact, for each class of imbalance functions, we need to have multiple instances with different basis functions while other parameters are kept fixed.

- **Balanced baselines:** To be able to compare CACCs against other round-robin CCs, we need to have a set of modular but completely uniform problems. While CACCs may outperform the traditional optimizers on imbalanced functions, they should not perform worse than them when the level of imbalance is zero or negligible. For the best practice, the class of balanced functions should have the same basis functions and number of components as imbalanced functions. Otherwise, it may introduce unwanted side effects that influence the results and conclusions.

- **Various imbalance scenarios:** Imbalance contributions can have different root causes such as unequal component coefficients, dimension sizes, or search landscapes. It is extremely important to cover all possible scenarios when comparing different algorithms. In addition, to study the effect of each source on the performance of the compared algorithms, the set should include problems with only one type of imbalance at a time.

- **Multiple imbalance level:** Only including problems with different types of imbalance is not enough. We need to be able to study the effect of each source of imbalance on a set of algorithms when the level of inequality varies.

- **Hybrid imbalance sources:** While including problems that only have one source of imbalance helps us to study them in an isolated environment, it is also of a great importance to include problems with multiple types of imbalance. For example, it would be interesting to investigate whether an algorithm invests more on optimizing components with higher dimensionality (*i.e.,* more complex subproblems) or larger

coefficients (*i.e.,* more influential subproblems). Different types of contribution imbalance may reinforce or weaken each others' effects.

- **Intraclass comparison:** We need to be able to compare the performance of a set of algorithms across different types and levels of contribution imbalance while other parameters are fixed. For example, given a fixed basis function, problem dimensionality, and number of components, we should be able to study the performance of an algorithm in three scenarios: when component coefficients are uniform, slightly different, and significantly nonuniform. To satisfy this condition, the number of subproblems, their sizes, coefficients, and basis functions should be kept fixed among different classes of problems as much as possible. Otherwise, a meaningful comparison between two or more categories of problems would be impossible.

- **Large sample size:** We need to have enough problem instances to be able to use proper statistical analysis. In particular, when performing significant tests on four or more algorithms, a large number of test cases should be available. Otherwise, drawing a confident conclusion can be challenging, if even possible.

In the following, we discuss a number of scenarios that cover all above points either individually or jointly. We also investigate the previously proposed benchmarking sets to see to what extent they address these scenarios. Then, we provide some guidelines to implement a comprehensive benchmark set that addresses all the outlined cases.

9.2 Contribution Imbalance Scenarios

In this section, we discuss six specific scenarios that will help us in analyzing CACCs in a level that was not possible before.

9.2.1 Case 1: Balanced Problems

No one can guarantee that all real-world partially separable functions consist of components with severe nonuniform contributions, although it is very likely. Therefore, it is of a great importance to examine the performance of contribution-aware algorithms on completely balanced problems to ensure that their budget allocation mechanism has no inverse effect when the imbalance is zero or negligible. In other words, CACCs should be able to perform at least equivalent to round-robin CCs in dealing with balanced functions.

Since CEC'13 does not cover balanced modular problems, most of the previous studies only analyzed the CACCs using imbalanced functions. In Chapter 8, we modified f_4–f_{11} from CEC'13 to create eight problems with uniform coefficients. However, the dimension sizes are still varying from one component to another. The only official benchmark set that have completely balanced functions is CEC'10 (f_{14}–f_{18}). Nevertheless, that set does not cover problems with nonuniform coefficients to compare with the balanced cases. To fill this gap, we include a subset of functions that each of which consists of components with equal sizes, uniform coefficients and the same basis functions to benchmark available algorithms.

9.2.2 Case 2: Nonuniform Coefficients

In this case, we aim to investigate the effect of unequal component coefficients on the efficiency of the algorithms. Multiplying components with nonuniform coefficients is the simplest way to introduce an imbalance effect into the test set. The contribution of

components, in this case, is optimizer-agnostic because the coefficients have no effect on the difficulty of solving the optimization problem.

Unlike in CEC'13 where coefficients are randomly generated, we propose to multiply each component with a coefficient chosen based on the component indices. For example, the coefficients can be defined as monotonic increasing or decreasing functions of indices. The systematic coefficient generation is more advantageous than random assignment (as in previously proposed benchmarks) because even before running the algorithms we know which component has the highest contribution. Note that this information will not be provided to the algorithm as the functions have to be presented as black-box problems to the optimizers.

The f_4–f_{11} from CEC'13 all have this type of contribution imbalance. However, the nonuniform coefficients are coupled with unequal dimensionality which together present a mixed effect on the final results. Since both the sizes and the coefficients are chosen randomly, it is practically difficult to extract a meaningful insight from the components selected by a CACC for further optimization.

In this study, we form a set of large-scale functions each consisting of a fixed number of components with the same dimensionality. Additionally, all components of a problem need to have the same basis function (we will study these factors separately). Therefore, nonuniform coefficients are the only source of the imbalanced contribution in this particular case.

To study the effect of this type of imbalance, we need to be able to control the degree of imbalance. A simple problem design to address this demand is to define $C_k = \alpha^k$ where α is a user-defined parameter that tunes the imbalance degree. For instance, $\alpha = 10$ introduces higher degree of imbalance than $\alpha = 2$. Note that the relation between the component index k and the value of C_k, which did not exist in the previous sets, helps us to easily identify the most and the least contributing components.

9.2.3 Case 3: Unequal Dimensionality

The problems that are proposed in Subsection 9.2.2 consists of components with unequal importance while all the components have the same search landscape and dimensionality. As a result, all subproblems have exactly the same level of difficulty from the subproblem solver's point of view. To include functions that consist of components with multiple levels of difficulty, we can easily change the dimensionality of subfunctions. Generally speaking, subproblems with higher dimensionality need more objective function calls to be solved. As a result, if the component sizes are defined as increasing functions of their indices, one can conclude that the subproblem with the largest index is the most difficult one.

In general, we expect effective CACCs to invest more budget on the subfunctions with larger sizes as they need more computational effort to be solved. In addition to their level of difficulty, such components may naturally produce larger objective values. As a result, a larger subproblem is generally the most contributing component. In practice, however, a CACC may invest more on smaller subproblem if it cannot see significant progress in the optimization of a very large and complex subproblem. In other words, an algorithm may observe that by spending a fraction of its budget on the smaller subfunction it can receive relatively larger improvement because the too large subproblems need a lot of effort to show any significant improvement. This is the reason we need to cover both nonuniform coefficients (which only affect the importance of components) and unequal dimensionality (which also affect the subproblems' complexity).

All imbalanced functions in CEC'13 (*i.e.,* f_4–f_{11}) reflect this type of imbalance. However, there are a few limitations. Firstly, the component sizes are selected arbitrarily and

there is no relation between the component sizes and indices. Secondly, unequal dimensionality is mixed with nonuniform coefficients which makes it difficult to study the effect of each of these factors separately. Thirdly, the number of subfunctions significantly varies from one instance to another. For example, f_4 consists of eight components whilst f_8 includes 20 subproblems. The number of candidate components can significantly affect the performance of budget allocation mechanism in contribution-aware algorithms. Therefore, it is recommended to fix this parameter or at least change it systematically (based on some experiment design).

9.2.4 Case 4: Heterogeneous Search Landscapes

In many real-world applications, the nature of each subfunction can differ significantly. For example, consider a multiobjective problem that consists of two or more conflicting objectives which is scalarized into one single-objective problem. It is very likely that the components of this single-objective problem present very different set of features. The features of the search landscape such as its modality, symmetry, and smoothness can significantly affect the performance of subproblem solvers. In addition, the performance of optimizers may be affected by such features. Therefore, we need to include this type of imbalanced components into our comprehensive set. Similar to problem proposed in Subsection 9.2.3 (Case 3), this imbalance source affects the difficulty of a subproblem. However, we cannot easily define the difficulty of a subproblem as a function of its index in this case (as opposed to the imbalance in dimensionality) because it also depends on the optimizer characteristics.

This type of imbalance is included in CEC'13 but in a very limited way. Each of f_4–f_7 consists of seven nonseparable and one fully separable subproblems where the basis functions for these two categories differ. Except for f_7 that two completely different basis functions are used (*i.e.,* Schwefel and Sphere), in the other three instances the basis functions are only slightly different. In fact, a rotated version of the same fully separable basis functions are used as nonseparable subproblems. Another limitation of these problems is their unequal dimension sizes. All separable components consist of 700 variables whilst other components are at most 100-dimensional. In addition, the nonseparable components are multiplied by some random coefficients whilst the separable subproblems always have a coefficient equal to 1. These coupling effects make it almost impossible to study the effect of heterogeneous landscape deeply and in an isolated environment.

9.2.5 Case 5: Conforming Imbalance Sources

All the aforementioned cases can help us to study the effect of each type of imbalanced contribution independently. These isolated factors provide great opportunity to deeply investigate each of them without the risk of being affected by other possible factors. However, these scenarios are not very likely in the real-world; there is a fair chance of having multiple causes contribute to making an objective function imbalanced. Therefore, it is reasonable to also investigate the cases that more than one source of imbalance exists.

In this particular case, we intend to study the mutual effect of two types of contribution imbalance: nonuniform coefficients and unequal dimensionality. In particular, we investigate the case that the largest components (which also should be the most difficult subfunctions) have the largest coefficients as well (to stress out their importance). Indeed, the term *conforming* is chosen for this type of mixed effect as the coefficients and dimension sizes have a positive correlation.

Although these two sources of imbalance seem to have similar result, they may cancel each others' effects in some special cases. For example, consider a complex basis function that has to be optimized using a weak optimizer [1]. As a result, the measured magnitude of the optimization progress dramatically drops as the dimensionality of the subproblem grows, especially when the given budget is limited (*e.g.,* short CC epochs or low population sizes). However, the large coefficients of these high-dimensional subproblems magnify even the smallest contributions towards solving the original problem. Therefore, the effect of different sources may not agree. This scenario is less likely when the adopted optimizer can achieve reasonable progress even on the large subproblems. Note that, conforming imbalanced sources are not systematically implemented in the previous large-scale benchmark problems as the component coefficients were randomly generated in CEC'13.

9.2.6 Case 6: Confronting Imbalance Sources

As opposed to conforming imbalance sources that discussed in Subsection 9.2.5, there could be cases that two or more imbalance sources to contradict each other. For example, consider having subproblems with different sizes when the smallest ones multiplied by the largest coefficients. In other words, the coefficients and dimension sizes have a negative correlation. In this case, an effective CACC should be able to easily pick the least complex but most influencing subproblems and solve them first.

The task may seem trivial, but in practice, it is still challenging. The first factor that contributes to the difficulty of the task is the differences between the dimension sizes. Indeed, the dimensionality of a subproblem can indirectly affect the objective value ranges especially in the case of high-dimensional subproblems. The cardinality of the range (*a.k.a.* co-domain) of many basis functions exponentially increases as the dimensionality of the function grows. This sudden change may cancel the effect of their small coefficients in practice.

Another source of difficulty in solving this case is knowing when to stop spending any more computational resource on the smallest subproblems (with the largest coefficients). Since these subfunctions significantly contribute towards the solving of the whole problem, a CACC may invest a considerable part of its budget towards solving them. However, it is very likely that these subfunctions being solved very soon (because of their small sizes) while the algorithm still tries to optimize them further as the history shows these subproblems used to be significantly contributing (refer to crossing moment concept in Chapter 8). An effective CACC should not fall into this trap and must be able to switch to other components as soon as the improvement received from the subproblems become superficial. Note that this is the first time that confronting imbalanced sources are systematically used in a large-scale optimization benchmark set. In the previous attempts, the coefficients are randomly generated regardless of the dimensionality of the subproblems.

9.3 Benchmark Design

In this section, we propose a set of large-scale problems for studying the issue of unequal contributions. First of all, we provide the general problem design principals that are common among all categories of problems. These points should be carefully taken into account to ensure we can address all the important features that are discussed in Section 9.1. Then, we define the mathematical symbols that will be used in the formal definitions. Finally,

[1] Both 'complex' and 'weak' can be subjective.

we explain each category of problems in details. Overall, these categories cover all the scenarios discussed in Section 9.2.

9.3.1 General Settings

We propose 40 large-scale optimization problems to deeply investigate the effectiveness of CACC algorithms in the six different scenarios that are discussed in Section 9.2. The set consists of eight categories each of which contains five problems. All problems are 1000-dimensional and each of them has exactly 10 components. None of these components has any interaction with any other component in the set. We deliberately fixed the dimensionality and number of subproblems to avoid any potential effect that these factors may impose on the outcome of the studies.

Overall, six basis functions are used in this set. All of the basis functions are borrowed from CEC'13 and we use exactly the same implementations. In all cases, the optimum values of problem instances are shifted to randomly chosen locations in order to avoid any bias towards the origin. In addition, the search landscapes are rotated to make each sub-function a nonseparable component. When applicable, the landscape symmetry is broken and some irregularities are added to the functions. Again, we follow the same symmetry breaking and irregularity addition approach as suggested in CEC'13 problems [Li et al. 2013a]. Since these basis and auxiliary functions have been used widely, their potentials and pitfalls are already known to the research community.

Although we use the same generators for creating random shift vectors and rotation matrices as proposed in CEC'13, we employ them in a slightly different direction. Previously, a single rotation matrix is used for all trials. Here, however, we use different shift vector and rotation matrix for each trial. For example, if an algorithm is applied to optimize a certain problem, the value of the global solution is changed to a different location for each different run. In addition, we use different rotation matrices in each run of an algorithm. As a result, even if an algorithm is executed several times using fixed parameter setting and the same initial population, the final solution would be totally different in each trial. Nevertheless, the same shift vectors and rotation matrices are used for trials with the same index. This means, the k^{th} runs of f_i and f_j share these vectors and matrices (as long as the dimensionalities of their components match).

The aforementioned approach of generating shift vectors and rotation matrices has several advantages. Firstly, it gives us an unlimited number of different instances of the same problem. This means, while the modularity, modality, and imbalance type and degree of all instances are the same, we still can have as many instances with different landscapes and global optima as needed. Therefore, since there is no limitation on the number of samples, a wide range of statistical tools can be used to assess the results of optimizers.

Secondly, different executions of optimizers with deterministic population initialization techniques will not produce the same result. This helps us to fairly compare them against optimizers which use stochastic initialization techniques (see Chapter 3).

Thirdly, we can pair and compare the trials of different algorithms using their index numbers. For example, assume we conducted a study to investigate the effect of the F value in DE/rand/1/bin using this benchmark set. We can fix all parameters (including initial population) and systematically change F value over several trials. Then, we can easily compare the k^{th} trials of DE/rand/1/bin with $F = 0.4$ and $F = 0.6$ as they are essentially the same problems. Without this particular feature, we would not be able to directly compare the k^{th} trials of two algorithms (or one algorithm with different settings) as they would represent different problems.

Finally, the ability to match trials with the same index is extremely beneficial in applying paired statistical analysis (*e.g.,* Friedman technique). Without this feature, we can only take some summary statistics (*e.g.,* mean or median) of all trials and pair them (based on problem index not the instance index). In CEC'13, for example, the old approach only gives us eight summary statistics to compare a set of algorithms. This makes the outcome of many statistical tests meaningless when the number of compared algorithms is more than three [García et al. 2010]. It is strongly advised that the number of samples (eight in the case of CEC'13) should be at least three times the number of treatments (*i.e.,* number of algorithms). In our approach, however, we can have as many samples as we want since every single trial serves as an independent sample for the statistical tests. We can also perform block analysis (using Friedman or Quade algorithms) as the set of instances of a particular problem form a block of independent samples [Sheskin 2003].

Another special consideration in designing this set of problems is to make the different category of problems comparable. To achieve this goal, we fixed the order of basis functions among all categories. Therefore, the problems that their indices modulo 5 are equal have the same basis function (the only exception is f_{26}-f_{30} which have heterogeneous search landscapes). This helps us to study the impact of imbalance sources and levels on each single basis function. For example, by comparing f_5, f_{10} and f_{15}, we can investigate the effects of three levels of unequal coefficients on Rosenbrock's problems. We will discuss this feature further in the following sections.

9.3.2 Formal Definition

In this section, we define the benchmark functions using a more formal language. Before presenting the problem definitions, we need to declare the mathematical notations that will be seen in the definitions. Note that similar to other parts of this dissertation, constants (*e.g.,* K) and variables (*e.g.,* k) are represented as uppercase and lowercase letters, respectively. The vectors are typeset in lowercase bold letters (*e.g.,* $\mathbf{x} = \langle x_1, \dots, x_D \rangle$), matrices in uppercase bold letters (*e.g.,* $\mathbf{X}$), and tensors (higher order matrices) in uppercase calligraphic font (*e.g.,* $\mathcal{X}$)[2]. The sets such as $\mathbb{N} = \{1, 2, \dots\}$ are represented using uppercase letters in blackboard style. To maximize the consistency with CEC'13 documentations, we used similar notations where possible. The following is a list of commonly used symbols and their definitions:

D : The dimensionality of the objective function which is fixed to 1000. In other words, $D = |\mathcal{D}|$ where $\mathcal{D} = \{1, \dots, 1000\}$ is the set of decision variable indices.

$\mathcal{D}_k$: A subset of $\mathcal{D}$ which denotes the variable indices of the k^{th} component. Its cardinality is indicated by $D_k = |\mathcal{D}_k|$.

K : The number of the subfunctions which is fixed to 10 in this set.

f_i : The i^{th} objective function in the set where $i \in \{1, \dots, 40\}$.

$f_{i,k}$: The k^{th} subfunction of the i^{th} objective function in the set where $i \in \{1, \dots, 40\}$ and $k \in \{1, \dots, 10\}$.

D_k : The dimensionality of the k^{th} subfunction which varies from one function to another. Unequal D_k values in a single problem causes imbalance. Table 91 presents the D_k values for all functions in the set.

[2]There are a few exceptions such as $\mathcal{D}$ that indicates the set of decision variables

Table 91: Subfunction dimension sizes. g, f, and k represent problem category, function number and subfunction index, respectively.

		k									
g	f	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
	1	100	100	100	100	100	100	100	100	100	100
	2	100	100	100	100	100	100	100	100	100	100
1	**3**	100	100	100	100	100	100	100	100	100	100
	4	100	100	100	100	100	100	100	100	100	100
	5	100	100	100	100	100	100	100	100	100	100
	6	100	100	100	100	100	100	100	100	100	100
	7	100	100	100	100	100	100	100	100	100	100
2	**8**	100	100	100	100	100	100	100	100	100	100
	9	100	100	100	100	100	100	100	100	100	100
	10	100	100	100	100	100	100	100	100	100	100
	11	100	100	100	100	100	100	100	100	100	100
	12	100	100	100	100	100	100	100	100	100	100
3	**13**	100	100	100	100	100	100	100	100	100	100
	14	100	100	100	100	100	100	100	100	100	100
	15	100	100	100	100	100	100	100	100	100	100
	16	50	50	50	100	100	100	100	150	150	150
	17	50	50	50	100	100	100	100	150	150	150
4	**18**	50	50	50	100	100	100	100	150	150	150
	19	50	50	50	100	100	100	100	150	150	150
	20	50	50	50	100	100	100	100	150	150	150
	21	25	25	50	50	75	75	100	150	200	250
	22	25	25	50	50	75	75	100	150	200	250
5	**23**	25	25	50	50	75	75	100	150	200	250
	24	25	25	50	50	75	75	100	150	200	250
	25	25	25	50	50	75	75	100	150	200	250
	26	100	100	100	100	100	100	100	100	100	100
	27	100	100	100	100	100	100	100	100	100	100
6	**28**	100	100	100	100	100	100	100	100	100	100
	29	100	100	100	100	100	100	100	100	100	100
	30	100	100	100	100	100	100	100	100	100	100
	31	25	25	50	50	75	75	100	150	200	250
	32	25	25	50	50	75	75	100	150	200	250
7	**33**	25	25	50	50	75	75	100	150	200	250
	34	25	25	50	50	75	75	100	150	200	250
	35	25	25	50	50	75	75	100	150	200	250
	36	25	25	50	50	75	75	100	150	200	250
	37	25	25	50	50	75	75	100	150	200	250
8	**38**	25	25	50	50	75	75	100	150	200	250
	39	25	25	50	50	75	75	100	150	200	250
	40	25	25	50	50	75	75	100	150	200	250

Table 92: Subfunction coefficients. g, f, and k represent problem category, function number and subfunction index, respectively.

		k									
g	f	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
	1	1	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	1	1	1	1	1	1
1	**3**	1	1	1	1	1	1	1	1	1	1
	4	1	1	1	1	1	1	1	1	1	1
	5	1	1	1	1	1	1	1	1	1	1
	6	2	4	8	16	32	64	128	265	512	1024
	7	2	4	8	16	32	64	128	265	512	1024
2	**8**	2	4	8	16	32	64	128	265	512	1024
	9	2	4	8	16	32	64	128	265	512	1024
	10	2	4	8	16	32	64	128	265	512	1024
	11	10	100	1000	10000	100000	1000000	10000000	100000000	1000000000	10000000000
	12	10	100	1000	10000	100000	1000000	10000000	100000000	1000000000	10000000000
3	**13**	10	100	1000	10000	100000	1000000	10000000	100000000	1000000000	10000000000
	14	10	100	1000	10000	100000	1000000	10000000	100000000	1000000000	10000000000
	15	10	100	1000	10000	100000	1000000	10000000	100000000	1000000000	10000000000
	16	1	1	1	1	1	1	1	1	1	1
	17	1	1	1	1	1	1	1	1	1	1
4	**18**	1	1	1	1	1	1	1	1	1	1
	19	1	1	1	1	1	1	1	1	1	1
	20	1	1	1	1	1	1	1	1	1	1
	21	1	1	1	1	1	1	1	1	1	1
	22	1	1	1	1	1	1	1	1	1	1
5	**23**	1	1	1	1	1	1	1	1	1	1
	24	1	1	1	1	1	1	1	1	1	1
	25	1	1	1	1	1	1	1	1	1	1
	26	1	1	1	1	1	1	1	1	1	1
	27	1	1	1	1	1	1	1	1	1	1
6	**28**	1	1	1	1	1	1	1	1	1	1
	29	1	1	1	1	1	1	1	1	1	1
	30	1	1	1	1	1	1	1	1	1	1
	31	2	4	8	16	32	64	128	265	512	1024
	32	2	4	8	16	32	64	128	265	512	1024
7	**33**	2	4	8	16	32	64	128	265	512	10240
	34	2	4	8	16	32	64	128	265	512	1024
	35	2	4	8	16	32	64	128	265	512	1024
	36	1024	512	256	128	64	32	16	8	4	2
	37	1024	512	256	128	64	32	16	8	4	2
8	**38**	1024	512	256	128	64	32	16	8	4	2
	39	1024	512	256	128	64	32	16	8	4	2
	40	1024	512	256	128	64	32	16	8	4	2

C_k : A predefined constant which is used as the coefficient of the k^{th} component. In cases that $C_k \neq 1$, it introduces imbalance effect. Table 92 provides the C_k values for all problems in the set.

$\overset{*}{\mathbf{x}}$: A randomly generated vector that serves as a shift vector to change the location

of the global optimum. Since the global optimum of all basis functions is originally located at the origin, the $\overset{*}{\mathbf{x}}$ that is used for shifting purposes becomes the new global optimum. In general, $\overset{*}{\mathbf{x}} \in \mathbb{R}^D$. To perform the shift operation, we need to calculate $f(\mathbf{x} - \overset{*}{\mathbf{x}})$ instead of $f(\mathbf{x})$. The same trials of different functions have the same shift vector while different trials of the same function use different shift vectors. More formally, $\forall i, j \in \{1, \ldots, 40\}\, \overset{*}{\mathbf{x}}^t_{f_i} = \overset{*}{\mathbf{x}}^{\grave{t}}_{f_j}$ *iff* $t = \grave{t}$. Therefore, $\overset{*}{\mathbf{x}}^t_{f_i} \neq \overset{*}{\mathbf{x}}^{\neg t}_{f_i}$

$\mathbf{R}$: A randomly generate orthogonal rotation matrix which is used to rotate the fitness landscape around various axes [Salomon 1995]. In this set, different trials of a function are rotated differently while the same trials of different functions share the same $\mathbf{R}$. In formal language, $\forall i, j \in \{1, \ldots, 40\}\, \mathbf{R}^t_{f_i} = \mathbf{R}^{\grave{t}}_{f_j}$ *iff* $t = \grave{t}$. Therefore, $\mathbf{R}^t_{f_i} \neq \mathbf{R}^{\neg t}_{f_i}$

T^α_{I} : An $\mathbb{R}^D \to \mathbb{R}^D$ transformation that creates smooth local irregularities to search landscape [Li et al. 2013a].

$$T^\alpha_{\mathrm{I}}(x_i) = \begin{cases} \exp\left(\log x_i + \alpha \sin(10 x_i \log x_i) + \alpha \sin(7.9 x_i \log x_i)\right) & \text{if } x_i > 0 \\ 1 & \text{if } x_i = 0 \\ -\exp\left(\log x_i + \alpha \sin(5.5 x_i \log |x_i|) + \alpha \sin(3.1 x_i \log |x_i|)\right) & \text{if } x_i < 0 \end{cases}$$

where the value of α is set to 0.049 as per suggestion made in [Li et al. 2013a] and the coefficients of x_i are selected following [Hansen et al. 2010]. In the above equation, $\exp(g(x))$ indicates $e^{g(x)}$ and $|x_i|$ means the absolute value of x_i.

T^α_{S} : An $\mathbb{R}^D \to \mathbb{R}^D$ transformation function that breaks the symmetry of the symmetric landscapes [Hansen et al. 2010].

$$T^\alpha_{\mathrm{S}}(x_i) = \begin{cases} x_i^{1+\beta \frac{i-1}{D-1} \sqrt{x_i}} & \text{if } x_i > 0 \\ x_i & \text{otherwise} \end{cases}$$

where α is set to 0.2 for all problems following CEC'13 [Li et al. 2013a].

$\mathbf{\Lambda}^\alpha$: A D_k-dimensional diagonal matrix with the diagonal elements $\lambda_{i,i} = \alpha^{\frac{i-1}{2(D_k-1)}}$. This matrix is used to create ill-conditioning [Hansen et al. 2010]. Following CEC'13, α is set to 10 for all problems.

b_j : Basis functions as building blocks of modular problems. All basis functions are borrowed from CEC'13 set. The basis functions are defines in Table 93. In the table, L_j and U_j indicate lower and upper bounds of the j^{th} basis function.

$\hat{\mathbf{x}}_k$: Denotes the k^{th} component after the application of shift, rotation, and one or more transformations. The transformations applied to a subsolution depend on the basis function that defines the landscape. Table 95 clarifies the transformations used to alter each basis function.

Category 1: Balanced Functions

This set corresponds to the balanced problems described in Section 9.2.1. All subproblems of a function in this category have the same size, coefficient, and basis function. More formally, $\forall k \in \{1, \ldots, K\}$ $D_k = 100$ and $C_k = 1$. We use b_j as the basis for all subfunctions of f_i where $1 \leq i \leq 5$ and $j = ((i-1) \bmod 5) + 1$.

Table 93: Definitions of basis functions. The L_j and U_j indicate the box constraints over b_j and e denotes the natural exponential function. Note that by definition, no optima exists outside of the box in any of the cases.

b_j	**Name**	**Definition**	L_j	U_j
b_0	Sphere	$\sum_{i=1}^{D} x_i^2$	-100	$+100$
b_1	Elliptic	$\sum_{i=1}^{D} 10^{6\frac{i-1}{D-1}} x_i^2$	-100	$+100$
b_2	Rastrigin's	$\sum_{i=1}^{D} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$	-5	$+5$
b_3	Ackley's	$e^1 - 20e^{(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2})} - e^{(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i))} + 20$	-32	$+32$
b_4	Schwefel's	$\sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_i\right)^2$	-100	$+100$
b_5	Rosenbrock's	$\sum_{i=1}^{D-1} \left[100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2\right]$	-100	$+100$

Category 2: Mild Nonuniform Coefficients

This category particularly addresses imbalance issue problems explained in Section 9.2.2. Therefore, $\forall k \in \{1, \ldots, K\}\ C_k = 2^k$. However, for $6 \leq i \leq 10$, all subfunctions of f_i have the same sizes and basis functions ($D_k = 100$ and $j = ((i-1) \bmod 5) + 1$).

Category 3: Severe Nonuniform Coefficients

This set is very similar to the category defined in Section 9.3.2 which also corresponds to unequal component coefficients case (see Section 9.2.2). The main difference between f_5–f_{10} and f_{11}–f_{15} is that the coefficients of the latter problems grow faster. More precisely, $\forall k \in \{1, \ldots, K\}\ C_k = 10^k$ whereas $D_k = 100$ and $j = ((i-1) \bmod 5) + 1$ for $11 \leq i \leq 15$.

Category 4: Mild Unequal Dimension Sizes

Components of functions in this category have slightly different dimension sizes. The dimensionality of the subproblems varies from 50 to 150. This set corresponds to the imbalance source discussed in Section 9.2.3. For $6 \leq i \leq 20$, all subfunctions of f_i have the same coefficients (*i.e.,* $C_k = 1$) and basis functions ($j = ((i-1) \bmod 5) + 1$). In this category, the dimension size of the k^{th} component is defines as:

$$D_k = \begin{cases} 50 & \text{if } 1 \leq k \leq 3, \\ 100 & \text{if } 4 \leq k \leq 7, \\ 150 & \text{if } 8 \leq k \leq 10. \end{cases}$$

Category 5: Severe Unequal Dimension Sizes

This category contains problem similar to the functions defined in Category 4, but with a more significant imbalance in the dimension sizes. The dimension sizes of the subproblems in this set vary from 25 to 250. Similar to the functions in Category 4, all components of a function in this category have the coefficients and basis functions. More formally, $\forall k \in \{1, \ldots, K\}\ C_k = 1$ and $\forall i \in \{21, \ldots, 25\}\ j = ((i-1) \bmod 5)+1$. The dimensionality

Table 94: Subfunctions' basis functions. g, f, and k represent problem category, function number and subfunction index, respectively. The numbers in the table indicate the indices of basis functions (see Table 93).

		k												k									
g	f	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	g	f	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
	1	1	1	1	1	1	1	1	1	1	1		**21**	1	1	1	1	1	1	1	1	1	1
	2	2	2	2	2	2	2	2	2	2	2		**22**	2	2	2	2	2	2	2	2	2	2
1	**3**	3	3	3	3	3	3	3	3	3	3	**5**	**23**	3	3	3	3	3	3	3	3	3	3
	4	4	4	4	4	4	4	4	4	4	4		**24**	4	4	4	4	4	4	4	4	4	4
	5	5	5	5	5	5	5	5	5	5	5		**25**	5	5	5	5	5	5	5	5	5	5
	6	1	1	1	1	1	1	1	1	1	1		**26**	0	0	1	1	2	2	3	3	4	4
	7	2	2	2	2	2	2	2	2	2	2		**27**	0	0	1	1	2	2	3	3	5	5
2	**8**	3	3	3	3	3	3	3	3	3	3	**6**	**28**	0	0	1	1	2	2	4	4	5	5
	9	4	4	4	4	4	4	4	4	4	4		**29**	0	0	1	1	3	3	4	4	5	5
	10	5	5	5	5	5	5	5	5	5	5		**30**	0	0	2	2	3	3	4	4	5	5
	11	1	1	1	1	1	1	1	1	1	1		**31**	1	1	1	1	1	1	1	1	1	1
	12	2	2	2	2	2	2	2	2	2	2		**32**	2	2	2	2	2	2	2	2	2	2
3	**13**	3	3	3	3	3	3	3	3	3	3	**7**	**33**	3	3	3	3	3	3	3	3	3	3
	14	4	4	4	4	4	4	4	4	4	4		**34**	4	4	4	4	4	4	4	4	4	4
	15	5	5	5	5	5	5	5	5	5	5		**35**	5	5	5	5	5	5	5	5	5	5
	16	1	1	1	1	1	1	1	1	1	1		**36**	1	1	1	1	1	1	1	1	1	1
	17	2	2	2	2	2	2	2	2	2	2		**37**	2	2	2	2	2	2	2	2	2	2
4	**18**	3	3	3	3	3	3	3	3	3	3	**8**	**38**	3	3	3	3	3	3	3	3	3	3
	19	4	4	4	4	4	4	4	4	4	4		**39**	4	4	4	4	4	4	4	4	4	4
	20	5	5	5	5	5	5	5	5	5	5		**40**	5	5	5	5	5	5	5	5	5	5

Table 95: Altered search landscapes. The j and k indicate the basis function and subproblem indices, respectively. For the sake of simplicity, $\mathbf{\Lambda}^{10}$, $T_{\mathrm{S}}^{0.2}$, and $T_{\mathrm{I}}^{0.049}$ are printed as $\mathbf{\Lambda}$, T_{S}, and T_{I}, respectively.

b_j	**Name**	**Transformations**
b_0	Sphere	$\hat{\mathbf{x}}_k = \mathbf{R} \times (\mathbf{x}_k - \overset{*}{\mathbf{x}}_k)$
b_1	Elliptic	$\hat{\mathbf{x}}_k = T_{\mathrm{I}}(\mathbf{R} \times (\mathbf{x}_k - \overset{*}{\mathbf{x}}_k))$
b_2	Rastrigin's	$\hat{\mathbf{x}}_k = \mathbf{\Lambda} \times T_{\mathrm{S}}(T_{\mathrm{I}}(\mathbf{R} \times (\mathbf{x}_k - \overset{*}{\mathbf{x}}_k)))$
b_3	Ackley's	$\hat{\mathbf{x}}_k = \mathbf{\Lambda} \times T_{\mathrm{S}}(T_{\mathrm{I}}(\mathbf{R} \times (\mathbf{x}_k - \overset{*}{\mathbf{x}}_k)))$
b_4	Schwefel's	$\hat{\mathbf{x}}_k = T_{\mathrm{S}}(T_{\mathrm{I}}(\mathbf{R} \times (\mathbf{x}_k - \overset{*}{\mathbf{x}}_k)))$
b_5	Rosenbrock's	$\hat{\mathbf{x}}_k = \mathbf{R} \times (\mathbf{x}_k - \overset{*}{\mathbf{x}}_k)$

of a subfunctions in this category is defines as:

$$D_k = \begin{cases} 25 & \text{if } k \in \{1,2\}, \\ 50 & \text{if } k \in \{3,4\}, \\ 75 & \text{if } k \in \{5,6\}, \\ 100 & \text{if } k = 7, \\ 150 & \text{if } k = 8, \\ 200 & \text{if } k = 9, \\ 250 & \text{if } k = 10. \end{cases}$$

Category 6: Functions with Heterogeneous Landscapes

This set which represents imbalance type discussed in Section 9.2.4 contains functions comprised of five different basis functions. Since each problem has 10 components, each basis function is used twice to construct a 1000-dimensional problem. Each of the problems in this category has a slightly different combination of basis functions. The basis function are shown in Table 94. All subproblems of a function in this set have the same sizes and coefficients ($\forall k \in \{1, \ldots, K\}\ D_k = 100$ and $C_k = 1$).

Category 7: Conforming Imbalance Sources

This category contains problems with multiple imbalance sources as discussed in Section 9.2.5. To preserve positive correlation between two sources of imbalance, we defined the subproblems size and coefficients as increasing functions of component indices such that $\forall k \in \{1, \ldots, K\}\ C_k = 2^k$ while the dimensionality of the subfunctions are defines as:

$$D_k = \begin{cases} 25 & \text{if } k \in \{1, 2\}, \\ 50 & \text{if } k \in \{3, 4\}, \\ 75 & \text{if } k \in \{5, 6\}, \\ 100 & \text{if } k = 7, \\ 150 & \text{if } k = 8, \\ 200 & \text{if } k = 9, \\ 250 & \text{if } k = 10. \end{cases}$$

All components of a function in this category have the same basis functions ($\forall i \in \{31, \ldots, 35\}\ j = ((i-1) \bmod 5) + 1$).

Category 8: Confronting Imbalance Sources

The functions in this set are very similar to the Category 7 problems except for the coefficients that are defined as a decreasing function of the component index. This set represents the mixed imbalance sources that discussed in Section 9.2.6. More precisely, $\forall k \in \{1, \ldots, K\}\ C_k = 2^{11-k}$ while the dimension sizes of the components are defined as:

$$D_k = \begin{cases} 25 & \text{if } k \in \{1, 2\}, \\ 50 & \text{if } k \in \{3, 4\}, \\ 75 & \text{if } k \in \{5, 6\}, \\ 100 & \text{if } k = 7, \\ 150 & \text{if } k = 8, \\ 200 & \text{if } k = 9, \\ 250 & \text{if } k = 10. \end{cases}$$

All subproblems of a function in this category have the same basis functions ($\forall i \in \{36, \ldots, 40\}\ j = ((i-1) \bmod 5) + 1$).

9.4 Numerical Experiments

In this section, we conduct a series of numerical experiments to demonstrate the potentials of the proposed problem set. The next subsections provide more information on the selected algorithms for benchmarking purposes, the setup of the experiments, the metrics we use to evaluate the performance of selected algorithms, the experimental results, and finally a detailed discussion about the obtained results.

9.4.1 Benchmarked Algorithms

We benchmark a round-robin CC and three variants of CBCC to exhibit the capabilities of the proposed test set in action. Among the selected algorithms, CC, CBCC1 and CBCC2 are discussed in Chapter 7 and CBCC3 is proposed in Chapter 8. All four algorithms use the same optimizer called SaNSDE which is a well-known self-adaptive Differential Evolution [Yang et al. 2008c]. When possible, the same parameter settings (*e.g.,* population size, initial cross-over rate, and epoch length) is used across all benchmarked algorithms (see Table 96).

9.4.2 Experiments Setup

Each CC technique is applied to each and every proposed problem for 25 trials. Since each trial corresponds to a different instance of a problem (due to different shift vectors and rotation matrices) there is no need to randomly reinitialize populations for each run of an experiment. Instead, to make sure the random initial population is not in favor of an algorithm, all of them start with exact same initial population. Since the problem instances are similar across all algorithms, we can directly compare the outcomes of different algorithms on any particular problem instance (see Section 9.3.1). Besides initial population, all common parameters are also fixed to the same values. Table 96 provides more details about the parameter settings.

Table 96: Parameter values that are used in the numerical experiments.

Parameter	Value	Description
N	50	Population size
δ_t	50	Epoch length
N_e	3.0e+6	Maximum function evaluation
μ_F	0.5	Mean of SaNSDE's scaling factor
σ_F	0.3	Standard deviation of SaNSDE's scaling factor
p_t	0.05	CBCC3's exploration probability

Measures and Metrics

We use two metrics to measure the performance of each algorithm on the proposed benchmarking problems. The first metric is the final objective value of the best-found solution after $3,000,000$ objective function evaluations. These values are reported in eight tables in the form of summary statistics (*i.e.,* median, mean, and standard deviation) of 25 independent trials. Since all proposed functions are minimization problems, a smaller average objective value indicates a better performing algorithm. The standard deviation values should also be taken into account as they can evidence the instability in the performance of an algorithm from one trial to another. Obviously, the lower the standard deviation the more confidence we have in the stability of the expected outcome of an algorithm.

One may conduct advanced statistical studies to measure the significance of the difference between the behavior of one or more CBCCs and the baseline to achieve a better understanding. In this section, however, we are only interested in illustrating the potentials of the benchmarks rather than deeply benchmarking the available algorithms. Therefore, we postpone more comprehensive studies to future work.

The percentage of budget allocated to solve each subproblem is another metric we use in this series of studies. Since the epoch length is fixed in all experiments, we simply

calculate the allocated budget to a component by counting the number of times it was selected during the optimization process. These numbers are reported as percentages of the total budget to be allocated while eliminating component switching overhead which is constant for all studied algorithms. This measurement is reported in the form of summary statistics in eight tables (each corresponds to a category of problems) as well as several illustrations. As seen in Chapter 8, this metric can reveal many insightful patterns that might not be observed solely from the final objective values.

9.4.3 Results and Discussions

This part consists of eight subsections each of which discusses the preliminary results of executing CC and CBCCs on one category of benchmark problems.

Category 1: Balanced Problems

Table 97 compares the performance of CC and three CBCC variants on the Category 1 problems. Since each of these problems consists of similar components with exactly the same level of contribution, we expect an effective CACC to perform close to a round-robin CC. In other words, the uniform resource allotment that is implemented in round-robin CCs is indeed the optimum budget allocation strategy for this category of problems. Therefore, CACCs should distribute the computational budget in almost the same way.

Table 97: Preliminary results of benchmarking a round-robin CC and three CBCCs on Category 1 (Balanced Problems). For each problem, smallest median values are written in **boldface** font to indicate the best performing algorithm.

		CC	CBCC1	CBCC2	CBCC3
f_1	Median	2.84E+07	2.85E+07	3.22E+10	**2.55E+07**
	Mean	2.83E+07	2.88E+07	3.21E+10	2.61E+07
	Std	2.81E+06	3.54E+06	1.64E+09	3.14E+06
f_2	Median	**3.51E+03**	3.66E+03	3.75E+03	3.59E+03
	Mean	3.50E+03	3.65E+03	3.75E+03	3.51E+03
	Std	1.50E+02	2.02E+02	1.40E+02	2.17E+02
f_3	Median	**2.13E+02**	**2.13E+02**	**2.13E+02**	**2.13E+02**
	Mean	2.13E+02	2.13E+02	2.13E+02	2.13E+02
	Std	1.22E-01	1.10E-01	1.25E-01	1.07E-01
f_4	Median	3.95E+06	4.07E+06	3.92E+06	**3.81E+06**
	Mean	3.95E+06	4.06E+06	4.06E+06	3.90E+06
	Std	6.94E+05	5.44E+05	6.56E+05	5.95E+05
f_5	Median	6.63E+03	6.72E+03	2.99E+11	**2.01E+03**
	Mean	8.72E+03	7.77E+03	2.59E+11	2.77E+03
	Std	5.30E+03	5.40E+03	1.10E+11	2.57E+03

As the numbers in Table 97 indicate, the CC and the three CBCC variants respond very similarly almost in all cases. The main exception here is the performance of CBCC2 on f_1 and f_5 which far worse than any other algorithm. In fact, the results obtained by CBCC2are on average four and eight orders of magnitude worse than other algorithms' outcomes on these particular problems. The unsatisfactory outcomes of CBCC2 can root in its greedy component selection strategy.

Table 98 summarizes the resource allocation performance of the CBCCs. We exclude the results from CC in this table because the round-robin strategy always distributes the shares uniformly. In such even budget assignment, each component will receive exactly 10% of available resources since all the proposed problems consists of 10 subproblems.

Table 98: The portion (in percentage) of allocated resources to solving each subproblem of Category 1 (Balanced) problems by CBCCs. Numbers are in $mean \pm std$ format.

	CBCC	k									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
f_1	**1**	$11_{\pm 3}$	$10_{\pm 3}$	$9_{\pm 2}$	$10_{\pm 3}$	$11_{\pm 4}$	$10_{\pm 3}$	$10_{\pm 3}$	$10_{\pm 3}$	$10_{\pm 3}$	$9_{\pm 4}$
	2	$12_{\pm 33}$	$12_{\pm 33}$	$8_{\pm 27}$	$8_{\pm 27}$	$16_{\pm 37}$	$16_{\pm 37}$	$4_{\pm 20}$	$8_{\pm 27}$	$8_{\pm 27}$	$8_{\pm 28}$
	3	$10_{\pm 1}$	$10_{\pm 1}$	$10_{\pm 2}$	$10_{\pm 2}$	$11_{\pm 2}$	$10_{\pm 2}$	$10_{\pm 2}$	$10_{\pm 2}$	$10_{\pm 2}$	$10_{\pm 3}$
f_2	**1**	$10_{\pm 2}$	$10_{\pm 2}$	$10_{\pm 3}$	$10_{\pm 3}$	$10_{\pm 3}$	$11_{\pm 4}$	$10_{\pm 3}$	$9_{\pm 0}$	$10_{\pm 2}$	$10_{\pm 4}$
	2	$10_{\pm 5}$	$9_{\pm 4}$	$9_{\pm 4}$	$10_{\pm 5}$	$9_{\pm 4}$	$10_{\pm 5}$	$8_{\pm 0}$	$11_{\pm 7}$	$11_{\pm 6}$	$12_{\pm 7}$
	3	$10_{\pm 2}$	$10_{\pm 2}$	$10_{\pm 2}$	$10_{\pm 2}$	$10_{\pm 2}$	$10_{\pm 3}$	$10_{\pm 2}$	$10_{\pm 2}$	$11_{\pm 2}$	$9_{\pm 3}$
f_3	**1**	$11_{\pm 3}$	$10_{\pm 3}$	$10_{\pm 2}$	$10_{\pm 3}$	$10_{\pm 2}$	$11_{\pm 3}$	$10_{\pm 2}$	$9_{\pm 1}$	$10_{\pm 3}$	$9_{\pm 4}$
	2	$11_{\pm 4}$	$9_{\pm 0}$	$9_{\pm 0}$	$10_{\pm 3}$	$10_{\pm 2}$	$10_{\pm 2}$	$10_{\pm 3}$	$10_{\pm 2}$	$11_{\pm 4}$	$10_{\pm 5}$
	3	$7_{\pm 3}$	$9_{\pm 9}$	$10_{\pm 6}$	$10_{\pm 10}$	$13_{\pm 13}$	$11_{\pm 8}$	$8_{\pm 6}$	$12_{\pm 10}$	$9_{\pm 9}$	$11_{\pm 10}$
f_4	**1**	$9_{\pm 2}$	$10_{\pm 3}$	$9_{\pm 2}$	$10_{\pm 3}$	$10_{\pm 3}$	$9_{\pm 2}$	$10_{\pm 3}$	$12_{\pm 4}$	$10_{\pm 3}$	$10_{\pm 4}$
	2	$10_{\pm 4}$	$10_{\pm 5}$	$11_{\pm 6}$	$9_{\pm 2}$	$10_{\pm 5}$	$10_{\pm 4}$	$11_{\pm 5}$	$9_{\pm 3}$	$10_{\pm 5}$	$9_{\pm 6}$
	3	$10_{\pm 3}$	$11_{\pm 3}$	$10_{\pm 3}$	$10_{\pm 2}$	$9_{\pm 2}$	$9_{\pm 3}$	$10_{\pm 2}$	$10_{\pm 3}$	$10_{\pm 2}$	$10_{\pm 3}$
f_5	**1**	$10_{\pm 3}$	$9_{\pm 0}$	$11_{\pm 3}$	$10_{\pm 3}$	$10_{\pm 3}$	$10_{\pm 3}$	$9_{\pm 0}$	$10_{\pm 3}$	$11_{\pm 4}$	$10_{\pm 5}$
	2	$3_{\pm 14}$	$12_{\pm 33}$	$4_{\pm 20}$	$8_{\pm 27}$	$12_{\pm 33}$	$24_{\pm 43}$	$12_{\pm 33}$	$4_{\pm 20}$	$16_{\pm 37}$	$4_{\pm 38}$
	3	$12_{\pm 10}$	$10_{\pm 10}$	$12_{\pm 8}$	$9_{\pm 6}$	$8_{\pm 6}$	$8_{\pm 6}$	$9_{\pm 8}$	$10_{\pm 10}$	$12_{\pm 11}$	$10_{\pm 12}$

The values in Table 98 indicate that in many cases, the CBCCs allocate around 10% of the resources to each component. However, CBCC2 functions very differently in two ways. Firstly, in the case of f_1 and f_5, the average allotted budget to some components are very large (up to 24% for $f_{5,6}$). This means some other subfunctions such as $f_{1,7}$ and $f_{5,1}$ (with less than 5% shares of budget) may not be optimized to a desired level. This can be the main contributing factor in the poor results of CBCC2 in dealing with f_1 and f_5. Secondly, the standard deviations obtained from CBCC2 are very large (up to 43% for $f_{5,6}$) whereas these numbers are around 3% for CBCC1 and CBCC3. This indicates the resource distribution may significantly change from one trial of CBCC2 to another. As a result, CBCC2 can be identified as the least stable algorithm among CBCC variants.

Figure 91 illustrates the median (in percentage) of the allocated resources to each component of f_1 and f_5 by CBCCs. The black dash line at 10% indicates the uniform resource allocation by the round-robin strategy. As the figure displays, CBCC1 and CBCC3 distribute the budget over all components almost uniformly. This trend is even more clear in f_1 as the blue and red lines are very close to the dashed line at 10%. The surprising observation from Figure 91 is that the green lines (indicating CBCC2 budget distribution) demonstrate a very uniform distribution but located around zero line. This kind of illustrations should not be misinterpreted. The graph does not suggest that CBCC2 allots the resources uniformly with not selecting any component (as the line is very close to zero line). The main reason is that the trend is calculated based on the median values which means a component may not be selected to be optimized in more than 50% of the trials. A scatter plot of all trial points should confirm this claim.

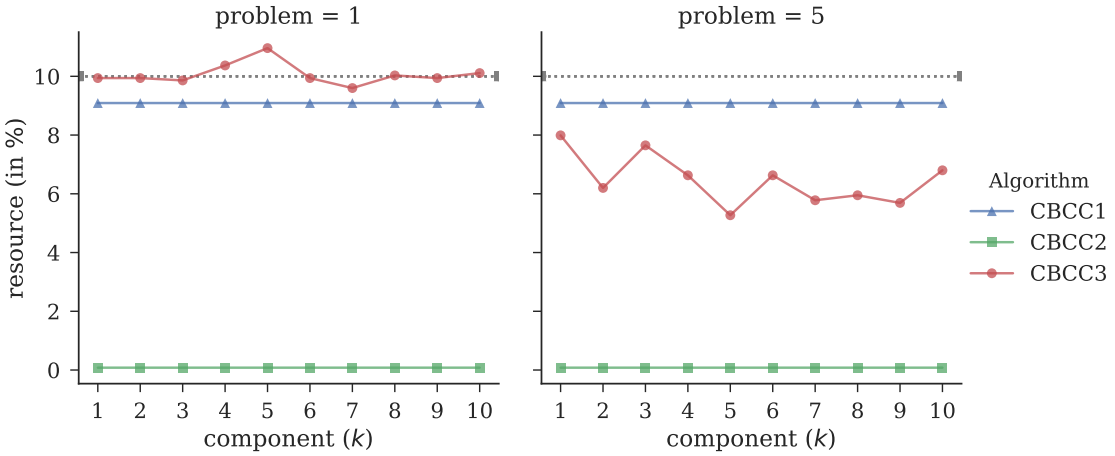


Figure 91: Median of allocated resources to components of f_1 and f_5. The black dash line at 10% indicates the uniform resource distribution.

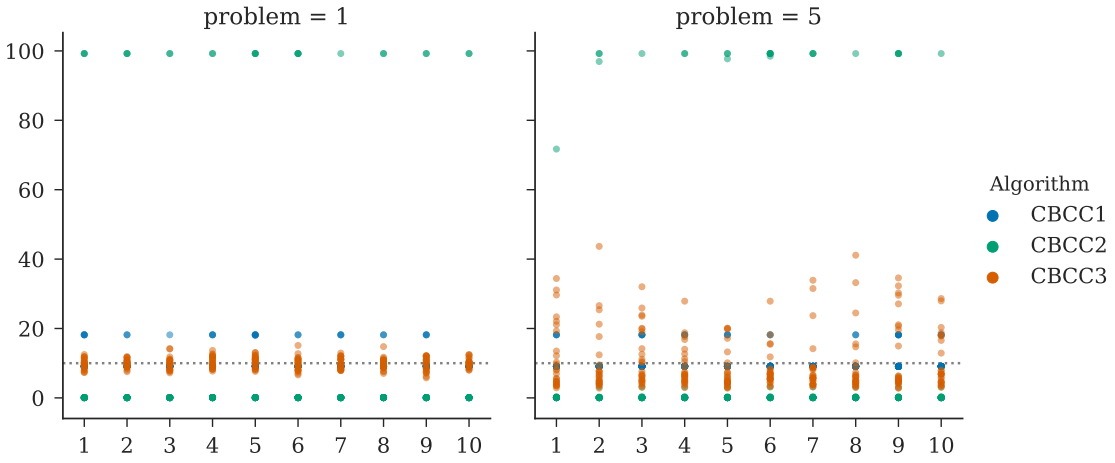


Figure 92: The allocated budget to components of f_1 and f_5. The black dash line at 10% indicates the uniform resource distribution.

Lots of green dots around zero in Figure 92 suggests once CBCC2 selects a component in exploitation phase, it rarely switches to any other components. This is the reason of zero medians in Figure 91 for CBCC2. It also means if CBCC2 chooses a component, it allocates almost all the budget to that particular component. This should explain the green dots around 100% and also the very large standard deviations in Table 98.

Comparing the performance of CC and CBCC variants on this category of problems supports the idea of including balanced problems with a uniform contribution in the benchmark set. In fact, this part shows some variants of CACCs such as CBCC1 can handle the balanced problems very effectively by uniformly distributing the resources among all components. However, some other techniques such as CBCC2 may fail in several ways. Therefore, in cases that we are not sure about the presence of imbalance in the contributions of the component, we definitely chose algorithms such as CBCC1 and CBCC3 over round-robin CC or CBCC2 because they can obtain similar or better results regardless of the absence or presence of the imbalance issue.

Category 2: Mild Nonuniform Coefficients

Table 99 compares the CCs and CBCCs results on the Category 2 problems. The table suggests that CBCC3 is the best algorithm when there is a minor nonuniformity in the coefficients of the subproblems. It also shows that CBCC1 provides slightly better results than the round-robin CC but cannot outperform CBCC3. CBCC2, however, only improves CC on f_7 whilst performs very poorly in f_6 and f_{10} cases. In fact, the quality of the solutions of CBCC2 is worse than the CC outcomes with three to eight orders of magnitude difference. The interesting fact here is that the outcome of CBCC2 on f_1 and f_5 (from Category 1) which have the same basis functions as f_6 and f_{10} were also very poor. Although all CBCCs and the round-robin CC in our experiments use exactly the same optimizer, the performance of CBCC2 is more sensitive to the search landscape features than other examined algorithms.

Table 99: Preliminary results of benchmarking a round-robin CC and three CBCCs on Category 2 (Mild Nonuniform Coefficients). For each problem, smallest median values are written in **boldface** font to indicate the best performing algorithm.

		CC	CBCC1	CBCC2	CBCC3
f_6	Median	5.26E+09	4.76E+09	3.78E+12	**3.07E+09**
	Mean	5.48E+09	4.84E+09	4.33E+12	3.05E+09
	Std	7.70E+08	6.77E+08	1.11E+12	3.12E+08
f_7	Median	7.36E+05	6.50E+05	5.88E+05	**5.01E+05**
	Mean	7.00E+05	6.50E+05	5.98E+05	5.02E+05
	Std	1.11E+05	4.61E+04	6.01E+04	5.78E+04
f_8	Median	4.36E+04	4.36E+04	4.36E+04	**4.35E+04**
	Mean	4.36E+04	4.36E+04	4.36E+04	4.35E+04
	Std	4.09E+01	4.67E+01	4.84E+01	6.11E+01
f_9	Median	7.16E+08	7.15E+08	8.24E+08	**7.08E+08**
	Mean	7.19E+08	6.92E+08	9.05E+08	6.64E+08
	Std	1.79E+08	2.06E+08	2.89E+08	1.83E+08
f_{10}	Median	8.14E+05	8.62E+05	3.65E+13	**2.73E+05**
	Mean	1.51E+06	1.37E+06	3.56E+13	3.25E+05
	Std	1.82E+06	1.69E+06	1.01E+13	1.15E+05

The numbers in Table 99 also show that CC and the studied CBCC variants perform similarly on f_8. This is another interesting observation as from Table 97 we know that these algorithms reach a similar result on f_3 which has the same basis function as f_8. This and the previous observations suggest that including a variety of basis functions in each category and matching similar basis functions among different categories is necessary. Note that these observations cannot be seen in the previously proposed benchmark sets as they lack such features.

Table 910 displays the distributions of resources over components of $f_6 - f_{10}$ allocated by CBCCs. Since the magnitude of subfunction coefficients is an increasing function of their indices (*i.e.*, $C_k = 2^k$), we expect that the 1^{st} and 10^{th} components to receive the smallest and the largest portions of the computational budget, respectively. As Table 910 shows, all CBCC variants selects the components with larger indices more often than the components with smaller indices. However, not all CBCCs distribute the resources

Table 910: The portion (in percentage) of allocated resources to solving each subproblem of Category 2 (Mild Nonuniform Coefficients) problems by CBCCs. Numbers are in $mean \pm std$ format.

	CBCC	C_k									
		$\mathbf{2^1}$	$\mathbf{2^2}$	$\mathbf{2^3}$	$\mathbf{2^4}$	$\mathbf{2^5}$	$\mathbf{2^6}$	$\mathbf{2^7}$	$\mathbf{2^8}$	$\mathbf{2^9}$	$\mathbf{2^{10}}$
f_6	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$10_{\pm 3}$	$17_{\pm 4}$
	2	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$4_{\pm 20}$	$28_{\pm 45}$	$68_{\pm 46}$
	3	$2_{\pm 0}$	$3_{\pm 0}$	$4_{\pm 1}$	$5_{\pm 1}$	$6_{\pm 1}$	$8_{\pm 2}$	$11_{\pm 2}$	$14_{\pm 2}$	$19_{\pm 3}$	$27_{\pm 4}$
f_7	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$18_{\pm 1}$
	2	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$26_{\pm 1}$
	3	$3_{\pm 0}$	$3_{\pm 1}$	$5_{\pm 1}$	$6_{\pm 1}$	$7_{\pm 2}$	$9_{\pm 2}$	$12_{\pm 3}$	$13_{\pm 4}$	$19_{\pm 4}$	$24_{\pm 5}$
f_8	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$12_{\pm 4}$	$16_{\pm 5}$
	2	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$12_{\pm 4}$	$16_{\pm 5}$
	3	$3_{\pm 0}$	$3_{\pm 0}$	$3_{\pm 0}$	$4_{\pm 1}$	$3_{\pm 1}$	$5_{\pm 4}$	$7_{\pm 5}$	$13_{\pm 13}$	$25_{\pm 17}$	$33_{\pm 18}$
f_9	**1**	$9_{\pm 0}$	$9_{\pm 2}$	$9_{\pm 2}$	$9_{\pm 2}$	$10_{\pm 3}$	$10_{\pm 3}$	$9_{\pm 2}$	$11_{\pm 4}$	$11_{\pm 4}$	$12_{\pm 5}$
	2	$8_{\pm 4}$	$8_{\pm 2}$	$8_{\pm 2}$	$11_{\pm 13}$	$8_{\pm 2}$	$10_{\pm 5}$	$10_{\pm 5}$	$16_{\pm 20}$	$10_{\pm 6}$	$12_{\pm 7}$
	3	$3_{\pm 0}$	$4_{\pm 0}$	$5_{\pm 1}$	$6_{\pm 1}$	$7_{\pm 1}$	$9_{\pm 2}$	$12_{\pm 3}$	$15_{\pm 3}$	$18_{\pm 4}$	$22_{\pm 5}$
f_{10}	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$10_{\pm 3}$	$17_{\pm 4}$
	2	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$4_{\pm 20}$	$95_{\pm 21}$
	3	$6_{\pm 8}$	$7_{\pm 8}$	$7_{\pm 6}$	$8_{\pm 9}$	$6_{\pm 4}$	$6_{\pm 3}$	$11_{\pm 9}$	$10_{\pm 8}$	$13_{\pm 12}$	$25_{\pm 13}$

similarly. For example, CBCC1 usually allocates 18% (or $100\frac{2K}{K+1}$ where $K = 10$) to the 10^{th} component (which is multiplied by the largest coefficient) and 9% or $100\frac{K}{K+1}$ to any other component. In some cases, it may spend slightly more on the second most contributing component (*i.e.,* 9^{th} subfunction) than the less-influencing subfunctions.

According to Table 910, CBCC2 invests up to 95% of the resources in solving the 10^{th} component and distributes the remaining budget almost evenly to the rest of subfunctions. There are two exceptions here; f_6 and f_{10}. In these cases, CBCC2 almost never selects the first seven or eight components for optimization. From Table 910 we recall that CBCC2 performs very poorly on these two functions. It seems never selecting most of the components, although with relatively smaller contributions, is the main reasons of weak performance of the CBCC2.

Among the examined strategies, the CBCC3 preserves a better balance between exploring all components and exploiting the most contributing components. According to Table 910, this strategy spends 22% – 33% of the budget to optimizing the most contributing component on average. It selects the second and third most important subproblems with 13% – 25% and 10% – 15% chances, respectively. Finally, CBCC3 distributes the rest of the resources to the less contributing components with a slight bias towards the subproblems with larger indices. Indeed, it allows the least contributing component (*i.e.,* the first subproblem with $C_1 = 2$) to consume 2% – 6% of the resources. Maintaining such balance between exploration and exploitation is the key factor in the superiority of CBCC3 over the earlier variants.

Another observation from Table 910 is that not only the average (*i.e.,* mean values) of the spent budget increase as the indices grow but also their standard deviations tend to increase as well. In CBCC1 and CBCC2, the standard deviations are almost zero for the

first seven subproblems (f_9 is an exception), whilst these values increase up to 46% for the last component. In the special case of CBCC2, the volatility in the allocated budget to 1^{st} and 10^{th} components of f_6 and f_{10} are extraordinary large. This is an interesting observation as we know CBCC2 performs very poorly in dealing with these two problems.

The results obtained from CBCC3 demonstrate a different pattern. The standard deviations of the allocated budget to the least contributing components by CBCC3 are slightly larger than CBCC1 and CBCC2 cases, although they are still negligible. However, these values are usually small especially in comparison with CBCC2 when it comes to selecting the most contributing components. These statistics affirm a relatively strong stability of the CBCC3 in resource allocation compared with the other two variants.

Category 3: Severe Nonuniform Coefficients

The problems in this category are very similar to Category 2 functions except the coefficients grow faster (*i.e.,* 10^k instead of 2^k). According to Table 911, CBCC2 and CBCC3 are the worst and best performing strategies, respectively. The table also suggests that CBCC1 improves round-robin CC in most cases but not as significantly as CBCC3 does. The very poor performance of CBCC2 in dealing with f_{11} and f_{15} resembles its failure in solving f_1, f_6, f_7, and f_{10}. Recall that all of these functions are built based on b_1 and b_5. This shows that CBCC2 is not particularly effective in solving Elliptic and Rosenbrock's problems. Further analyses need to be done to investigate the root cause of this shortcoming.

Table 911: Preliminary results of benchmarking a round-robin CC and three CBCCs on Category 3 (Severe Nonuniform Coefficients). For each problem, smallest median values are written in **boldface** font to indicate the best performing algorithm.

		CC	CBCC1	CBCC2	CBCC3
f_{11}	Median	2.75E+16	1.69E+16	3.82E+18	**5.25E+15**
	Mean	2.83E+16	1.78E+16	5.03E+18	5.45E+15
	Std	5.96E+15	4.20E+15	5.93E+18	1.14E+15
f_{12}	Median	4.13E+12	2.98E+12	2.48E+12	**2.05E+12**
	Mean	4.05E+12	2.78E+12	2.47E+12	2.04E+12
	Std	4.59E+11	5.20E+11	3.60E+11	3.18E+11
f_{13}	Median	2.37E+11	**2.36E+11**	**2.36E+11**	**2.36E+11**
	Mean	2.37E+11	2.36E+11	2.36E+11	2.36E+11
	Std	2.89E+08	3.39E+08	4.21E+08	4.90E+08
f_{14}	Median	4.85E+15	4.59E+15	3.73E+15	**2.70E+15**
	Mean	4.71E+15	4.35E+15	4.40E+15	2.37E+15
	Std	1.55E+15	1.59E+15	3.08E+15	1.55E+15
f_{15}	Median	1.73E+12	1.91E+12	3.72E+19	**1.58E+12**
	Mean	6.20E+12	3.18E+12	3.60E+19	3.22E+12
	Std	1.10E+13	2.93E+12	5.75E+18	2.89E+12

Another observation from Table 911 is that the variations in final objective values of all algorithms are larger than the values in Table 97–99. Since the range of objective values for this category of problems are generally larger than the other two categories (because of the large multiplied coefficients), the standard deviations are expected to be

larger as well. By comparing the standard deviations of the results from CBCC3 with the other algorithms, we observe that this algorithm is not the most steady one (at least in terms of the objective value of the final solutions) anymore. In other words, the variations in the results obtained by CBCC3 increased faster than CC and CBCC2 as a result of increasing the nonuniformity in the coefficients.

Table 912: The portion (in percentage) of allocated resources to solving each subproblem of Category 3 (Severe Nonuniform Coefficients) problems by CBCCs. Numbers are in $mean \pm_{std}$ format.

	CBCC	C_k									
		$\mathbf{10^1}$	$\mathbf{10^2}$	$\mathbf{10^3}$	$\mathbf{10^4}$	$\mathbf{10^5}$	$\mathbf{10^6}$	$\mathbf{10^7}$	$\mathbf{10^8}$	$\mathbf{10^9}$	$\mathbf{10^{10}}$
f_{11}	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$18_{\pm 1}$
	2	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$4_{\pm 20}$	$95_{\pm 21}$
	3	$1_{\pm 0}$	$1_{\pm 0}$	$1_{\pm 0}$	$1_{\pm 0}$	$1_{\pm 0}$	$2_{\pm 0}$	$4_{\pm 1}$	$8_{\pm 2}$	$21_{\pm 4}$	$61_{\pm 5}$
f_{12}	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$18_{\pm 1}$
	2	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$26_{\pm 1}$
	3	$2_{\pm 0}$	$2_{\pm 0}$	$2_{\pm 0}$	$2_{\pm 0}$	$2_{\pm 0}$	$3_{\pm 1}$	$5_{\pm 3}$	$10_{\pm 3}$	$25_{\pm 10}$	$44_{\pm 11}$
f_{13}	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$18_{\pm 1}$
	2	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$18_{\pm 1}$
	3	$3_{\pm 0}$	$3_{\pm 0}$	$3_{\pm 0}$	$3_{\pm 0}$	$3_{\pm 0}$	$3_{\pm 0}$	$3_{\pm 0}$	$4_{\pm 1}$	$12_{\pm 14}$	$61_{\pm 15}$
f_{14}	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$10_{\pm 3}$	$9_{\pm 0}$	$10_{\pm 3}$	$10_{\pm 3}$	$16_{\pm 4}$
	2	$7_{\pm 3}$	$7_{\pm 3}$	$7_{\pm 3}$	$7_{\pm 3}$	$7_{\pm 3}$	$7_{\pm 3}$	$11_{\pm 15}$	$8_{\pm 4}$	$18_{\pm 25}$	$19_{\pm 26}$
	3	$2_{\pm 0}$	$2_{\pm 0}$	$2_{\pm 0}$	$2_{\pm 0}$	$3_{\pm 0}$	$3_{\pm 1}$	$5_{\pm 2}$	$11_{\pm 5}$	$22_{\pm 8}$	$46_{\pm 9}$
f_{15}	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$18_{\pm 1}$
	2	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$99_{\pm 1}$
	3	$1_{\pm 0}$	$1_{\pm 0}$	$1_{\pm 0}$	$1_{\pm 0}$	$2_{\pm 1}$	$2_{\pm 1}$	$7_{\pm 11}$	$8_{\pm 8}$	$13_{\pm 18}$	$63_{\pm 19}$

Table 912 presents the allotted budget to each component of f_{11}–f_{15} by CBCCs. According to the table, CBCC1 successfully identifies the most contributing component and spends twice the amount of resources to it as allocates to other components. The very small standard deviations (almost zero) indicates that the CBCC1 is consistent in dealing with extremely nonuniform coefficients. CBCC2 functions very similar to CBCC1 in most cases except f_{11} and f_{15}. In these two cases, CBCC2 assigns almost all the resources to the component with the largest coefficient. Therefore, the other components are almost never selected. This aggressive distribution of resources is very close to the CBCC2 performance on f_6 and f_{10} which consist of components with the same basis functions. This finding shows that including problems with the same landscape but different levels of imbalance in the benchmark set is extremely important.

According to Table 912, the CBCC3 performance on Category 3 problems is comparable with its behavior in dealing with Category 2 problems. The only noticeable difference is that the slopes of resource distributions are steeper here. In fact, it assigns about 3% of the budget to the first six components, 4%–11% to the 7^{th} and 8^{th} components, up to 25% to the second most contributing component, and 44%–63% to the component with the largest coefficient. In comparison with the other CBCCs, the CBCC3 spends fewer resources on the least contributing components except in the cases of f_{11} and f_{15}. In general, it spends more resources on the second and third most important subproblems (*i.e.,* 8^{th} and 9^{th} components) than any other CBCC variants. Similar to Category 2, the

standard deviations of CBCC3 results on budget allotment increase as the subproblems' indices grow.

Figure 93 and 94 compare the budget allocation patterns of CBCCs on the Category 1–3 of problems. Note that each row of the subplots corresponds to one category of problems whereas all functions in a column are built using similar basis functions. Therefore, the only difference between the subfigures in a row is their basis functions whilst the difference between the functions plotted in subplots forming a column is their degree of coefficients nonuniformity (*i.e.,* level of imbalance).

The main observation from Figure 93 is that in average all CBCCs can find the most contributing components successfully. As a result, in the case of imbalanced contributions (*i.e.,* the second and third rows), the median of the spent budget on subproblems increases as the components' indices grow. The visible difference between the second and the third rows is the slope of the red line that represents the median of allocated resources to subproblems by CBCC3. In general, this slope is steeper in the third row (corresponds to severe nonuniformity) than the second row (mild coefficient imbalance).

The key finding from Figure 94 is that not only the magnitude of the spend budget increases as the coefficients enlarge but also the variations in allocated budget grow as well. The figure also confirms that CBCC2 behavior is less steady in comparison with the others as it shows very large volatility in spending the budget from one trial to another.

Category 4: Mild Unequal Dimension Sizes

The statistics presented in Table 913 confirms that the behavior of CBCCs on problems with moderate uneven component sizes is comparable with their performance on functions with mild nonuniform coefficients (provided in Table 99). The outcomes of all techniques are close to each other in most cases. This fact is more evident in f_{18} which CC and all CBCC variants return virtually the same solutions. Interestingly, there is a bold synergy between the effectiveness of these algorithms on f_8 and f_{18} (see Table 99). It seems the main reason is that both problems are built based on Ackley's function (*i.e.,* b_3) since they do not have any other feature in common.

In general, Table 913 suggests that the CBCC2 and CBCC3 are still the worst and best performing algorithms, respectively. The gap between the functionality of these strategies is more visible in f_{16} and f_{20} cases where CBCC2 performs very poorly in comparison with the others. This finding is consistent with the observations from Table 99 and 911; CBCC2 is not very efficient in dealing with Elliptic and Rosenbrock's functions.

As Table 914 presents, CBCC1 usually (except in case f_{18}) spends around 9% of the resources on each of subproblem with 50 or 100 decision variables and about 11%–13% on each of the subproblems with 150 dimensions. CBCC2 behaves similar to CBCC1 on f_{12} and f_{19} only with a larger volatility. In f_{16} and f_{20} cases where CBCC2 is very inefficient, it allocates almost zero resource to the smallest components, a small portion (usually about 4%–8%) to each middle-sized subproblem, and a large percentage (from 20% up to 40%) to the largest subproblems. While this confirms that CBCC2 finds the most contributing components correctly, spending almost nothing to solve small-sized subproblems seems to be the cause of the issue.

According to Table 914, CBCC3 almost always (except in f_{18} case) allocates 4%–8% of the resources to each of the 50-dimensional components, around 10% to middle-sized subproblems, and 11%–21% to the largest components. Therefore, we can conclude that distribution obtained by the CBCC3 sits somewhere between allocation strategy of the CBCC1 and CBCC2.

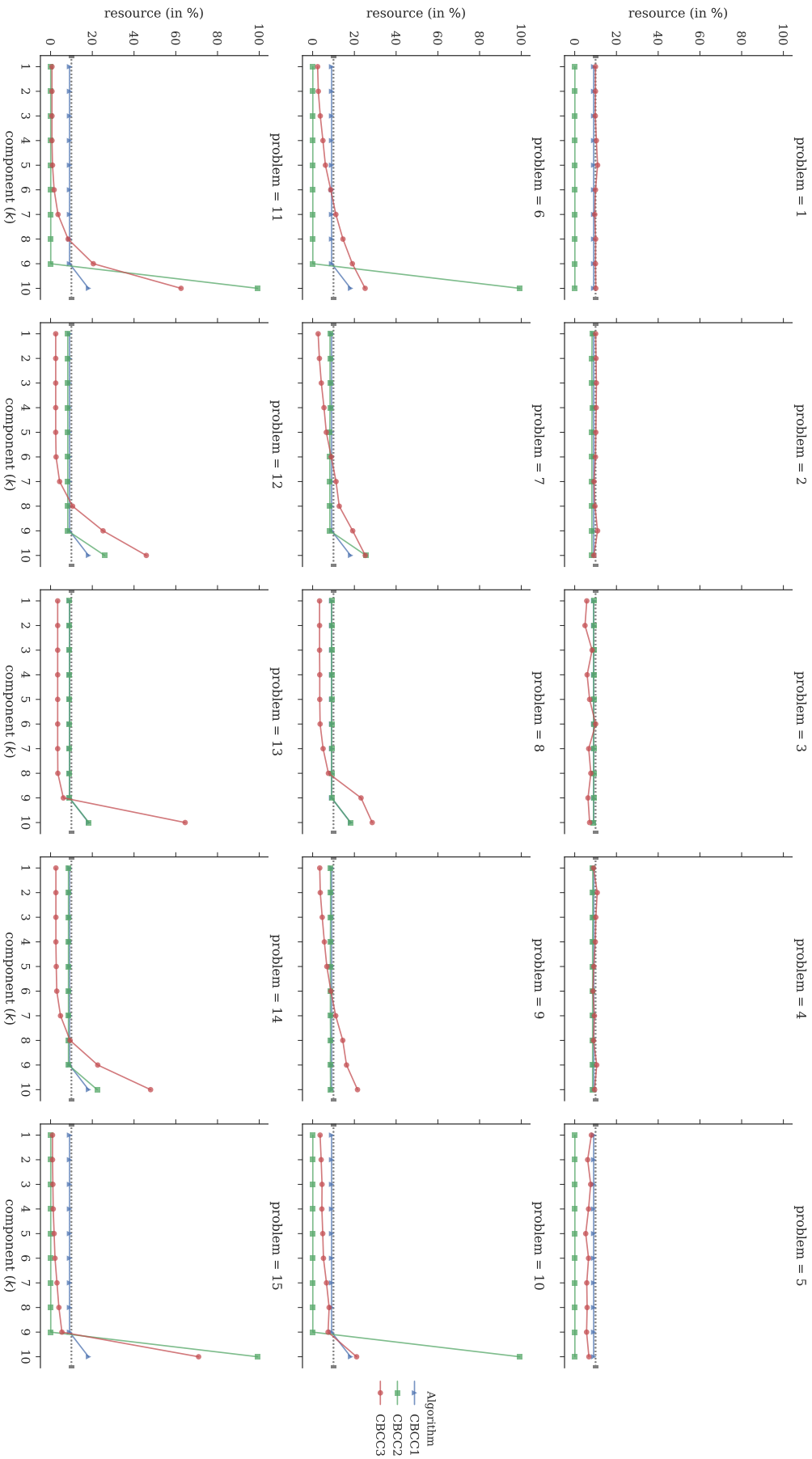


Figure 93: The median allocated budget to components of f_1 – f_{15}. The black dash line at 10% indicates the uniform resource distribution.

Figure 94: The allocated budget to components of f_1 – f_{15}. The black dash line at 10% indicates the uniform resource distribution.

Table 913: Preliminary results of benchmarking a round-robin CC and three CBCCs on Category 4 (Mild Unequal Dimension Sizes). For each problem, smallest median values are written in **boldface** font to indicate the best performing algorithm.

		CC	CBCC1	CBCC2	CBCC3
f_{16}	Median	4.27E+07	4.03E+07	3.24E+10	**2.95E+07**
	Mean	4.35E+07	3.98E+07	3.27E+10	2.98E+07
	Std	6.04E+06	4.30E+06	2.52E+09	2.62E+06
f_{17}	Median	3.75E+03	3.84E+03	3.87E+03	**3.74E+03**
	Mean	3.75E+03	3.82E+03	3.83E+03	3.67E+03
	Std	2.21E+02	1.52E+02	2.60E+02	2.59E+02
f_{18}	Median	**2.13E+02**	**2.13E+02**	**2.13E+02**	**2.13E+02**
	Mean	2.13E+02	2.13E+02	2.13E+02	2.13E+02
	Std	1.43E-01	1.37E-01	1.24E-01	1.38E-01
f_{19}	Median	4.36E+06	4.70E+06	5.00E+06	**4.01E+06**
	Mean	4.35E+06	4.50E+06	5.25E+06	4.09E+06
	Std	6.24E+05	7.81E+05	1.85E+06	9.81E+05
f_{20}	Median	6.59E+03	3.87E+03	3.12E+11	**1.93E+03**
	Mean	6.02E+03	5.48E+03	3.15E+11	2.41E+03
	Std	3.67E+03	4.03E+03	3.89E+10	1.70E+03

Table 914: The portion (in percentage) of allocated resources to solving each subproblem of Category 4 (Mild Unequal Dimension Sizes) problems by CBCCs. D_kn represents the dimensionality of k^{th} subproblem. Numbers are in $mean \pm std$ format.

	CBCC	D_k									
		50	**50**	**50**	**100**	**100**	**100**	**100**	**150**	**150**	**150**
f_{16}	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 2}$	$9_{\pm 2}$	$12_{\pm 4}$	$12_{\pm 4}$	$12_{\pm 5}$
	2	$0_{\pm 0}$	$0_{\pm 0}$	$4_{\pm 20}$	$16_{\pm 37}$	$0_{\pm 0}$	$8_{\pm 27}$	$8_{\pm 27}$	$24_{\pm 43}$	$20_{\pm 40}$	$20_{\pm 41}$
	3	$4_{\pm 1}$	$4_{\pm 1}$	$4_{\pm 1}$	$10_{\pm 2}$	$9_{\pm 1}$	$9_{\pm 1}$	$10_{\pm 2}$	$17_{\pm 2}$	$17_{\pm 3}$	$16_{\pm 4}$
f_{17}	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 1}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$12_{\pm 4}$	$12_{\pm 4}$	$12_{\pm 5}$
	2	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$9_{\pm 4}$	$8_{\pm 0}$	$8_{\pm 0}$	$15_{\pm 9}$	$14_{\pm 9}$	$13_{\pm 10}$
	3	$7_{\pm 1}$	$8_{\pm 2}$	$7_{\pm 1}$	$11_{\pm 2}$	$10_{\pm 2}$	$10_{\pm 2}$	$11_{\pm 2}$	$12_{\pm 3}$	$12_{\pm 3}$	$12_{\pm 4}$
f_{18}	**1**	$12_{\pm 4}$	$12_{\pm 4}$	$13_{\pm 4}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 1}$
	2	$13_{\pm 5}$	$10_{\pm 3}$	$12_{\pm 4}$	$9_{\pm 2}$	$9_{\pm 0}$	$9_{\pm 0}$	$10_{\pm 2}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 1}$
	3	$13_{\pm 8}$	$12_{\pm 8}$	$14_{\pm 12}$	$10_{\pm 10}$	$9_{\pm 7}$	$12_{\pm 9}$	$10_{\pm 9}$	$6_{\pm 4}$	$5_{\pm 3}$	$9_{\pm 4}$
f_{19}	**1**	$9_{\pm 2}$	$9_{\pm 0}$	$9_{\pm 2}$	$10_{\pm 3}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 2}$	$12_{\pm 4}$	$12_{\pm 4}$	$11_{\pm 5}$
	2	$8_{\pm 4}$	$7_{\pm 3}$	$9_{\pm 8}$	$11_{\pm 16}$	$8_{\pm 5}$	$8_{\pm 4}$	$8_{\pm 5}$	$15_{\pm 18}$	$13_{\pm 18}$	$12_{\pm 19}$
	3	$6_{\pm 1}$	$6_{\pm 1}$	$6_{\pm 1}$	$10_{\pm 3}$	$12_{\pm 3}$	$10_{\pm 2}$	$9_{\pm 3}$	$13_{\pm 5}$	$13_{\pm 4}$	$15_{\pm 5}$
f_{20}	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$10_{\pm 3}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$13_{\pm 5}$	$11_{\pm 4}$	$11_{\pm 5}$
	2	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$4_{\pm 20}$	$4_{\pm 20}$	$4_{\pm 20}$	$0_{\pm 0}$	$40_{\pm 50}$	$28_{\pm 45}$	$20_{\pm 46}$
	3	$6_{\pm 5}$	$5_{\pm 5}$	$6_{\pm 5}$	$8_{\pm 12}$	$11_{\pm 12}$	$11_{\pm 11}$	$8_{\pm 9}$	$21_{\pm 17}$	$11_{\pm 4}$	$12_{\pm 5}$

The resource distributions in the case of f_{18} is the most surprising observation from Table 914. As the table shows, all CBCC variants spend more effort to solve the smallest subproblems rather than the largest ones. This finding does not suggest that CBCCs are incapable of identifying the most contributing component. Instead, it shows that high-dimensional Ackley's subproblems are too difficult for the hired optimizer to progress in a single epoch. In other words, this particular optimizer can enhance objective function to a greater extent if it spends the fixed given budget on smaller Ackley's subproblems (as they are less complex) than the larger ones. Therefore, all three strategies select the 50-dimensional components more often than 150-dimensional subproblems. Perhaps increasing the epoch length (number of generations spent on optimizing a selected subproblem) or using a more powerful optimizer may change the result. We postpone this tweaks to future studies.

Category 5: Severe Unequal Dimension Sizes

Table 915 exhibits very similar results as the Table 913. The CBCC3 appears to be the best in virtually all cases, CBCC2 performance is very low in Elliptic and Rosenbrock's problems (*i.e.,* f_{21} and f_{25}), and all strategies obtain similar results on Ackley's problem (*i.e.,* f_{23}).

Table 915: Preliminary results of benchmarking a round-robin CC and three CBCCs on Category 5 (Severe Unequal Dimension Sizes). For each problem, smallest median values are written in **boldface** font to indicate the best performing algorithm.

		CC	CBCC1	CBCC2	CBCC3
f_{21}	Median	8.46E+07	7.02E+07	3.00E+10	**3.92E+07**
	Mean	8.39E+07	7.20E+07	3.14E+10	3.98E+07
	Std	1.13E+07	8.84E+06	4.55E+09	3.99E+06
f_{22}	Median	4.44E+03	4.25E+03	4.15E+03	**4.06E+03**
	Mean	4.34E+03	4.24E+03	4.10E+03	4.02E+03
	Std	3.19E+02	2.55E+02	3.35E+02	2.85E+02
f_{23}	Median	**2.12E+02**	**2.12E+02**	**2.12E+02**	**2.12E+02**
	Mean	2.12E+02	2.10E+02	2.12E+02	2.12E+02
	Std	1.49E-01	5.48E+00	2.46E-01	4.08E-01
f_{24}	Median	4.86E+06	**4.15E+06**	6.21E+06	5.07E+06
	Mean	4.80E+06	4.60E+06	1.31E+07	4.57E+06
	Std	1.27E+06	1.24E+06	3.65E+07	1.43E+06
f_{25}	Median	5.25E+03	7.35E+03	3.20E+11	**1.96E+03**
	Mean	6.50E+03	7.95E+03	3.22E+11	3.77E+03
	Std	4.15E+03	4.44E+03	4.51E+10	3.25E+03

The main difference between Table 913 and 915 is that in the latter case, CBCC1 appears to overcome CBCC3 on f_{24} according to the median of the objective values of the best solutions. However, the mean values suggest that CBCC3 is slightly more efficient than CBCC1 in this particular case.

The statistics in Table 916 shares lots of similarities with Table 914. The main pattern is that all CBCCs spend much more on solving the larger subproblems than the smaller ones, except in the case f_{23} (the Ackley's function). Again, CBCC2 spends almost nothing

Table 916: The portion (in percentage) of allocated resources to solving each subproblem of Category 5 (Severe Unequal Dimension Sizes) problems by CBCCs. D_k represents the dimensionality of k^{th} subproblem. Numbers are in $mean \pm std$ format.

	CBCC	D_k									
		25	**25**	**50**	**50**	**75**	**75**	**100**	**150**	**200**	**250**
f_{21}	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$10_{\pm 3}$	$11_{\pm 4}$	$16_{\pm 5}$
	2	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$8_{\pm 27}$	$8_{\pm 27}$	$24_{\pm 43}$	$60_{\pm 44}$
	3	$2_{\pm 0}$	$2_{\pm 1}$	$4_{\pm 1}$	$4_{\pm 0}$	$6_{\pm 1}$	$6_{\pm 1}$	$9_{\pm 1}$	$15_{\pm 2}$	$23_{\pm 3}$	$31_{\pm 4}$
f_{22}	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$11_{\pm 3}$	$17_{\pm 4}$
	2	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$9_{\pm 4}$	$27_{\pm 5}$
	3	$5_{\pm 1}$	$5_{\pm 1}$	$8_{\pm 1}$	$8_{\pm 1}$	$10_{\pm 2}$	$10_{\pm 2}$	$12_{\pm 2}$	$14_{\pm 2}$	$13_{\pm 3}$	$15_{\pm 4}$
f_{23}	**1**	$13_{\pm 4}$	$13_{\pm 4}$	$10_{\pm 3}$	$9_{\pm 1}$	$9_{\pm 1}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 1}$
	2	$13_{\pm 5}$	$14_{\pm 7}$	$9_{\pm 2}$	$10_{\pm 3}$	$9_{\pm 0}$	$9_{\pm 1}$	$9_{\pm 1}$	$9_{\pm 1}$	$9_{\pm 1}$	$9_{\pm 2}$
	3	$13_{\pm 7}$	$15_{\pm 12}$	$13_{\pm 10}$	$10_{\pm 7}$	$7_{\pm 6}$	$10_{\pm 10}$	$11_{\pm 11}$	$8_{\pm 6}$	$7_{\pm 6}$	$6_{\pm 7}$
f_{24}	**1**	$9_{\pm 0}$	$9_{\pm 2}$	$10_{\pm 3}$	$9_{\pm 2}$	$10_{\pm 3}$	$9_{\pm 0}$	$9_{\pm 2}$	$11_{\pm 4}$	$10_{\pm 3}$	$13_{\pm 4}$
	2	$8_{\pm 5}$	$9_{\pm 5}$	$8_{\pm 4}$	$7_{\pm 3}$	$11_{\pm 12}$	$8_{\pm 4}$	$7_{\pm 3}$	$10_{\pm 7}$	$9_{\pm 6}$	$22_{\pm 7}$
	3	$3_{\pm 1}$	$3_{\pm 1}$	$6_{\pm 1}$	$6_{\pm 1}$	$9_{\pm 2}$	$9_{\pm 2}$	$10_{\pm 3}$	$14_{\pm 5}$	$17_{\pm 5}$	$24_{\pm 6}$
f_{25}	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 2}$	$10_{\pm 3}$	$17_{\pm 4}$
	2	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$24_{\pm 43}$	$75_{\pm 44}$
	3	$4_{\pm 4}$	$4_{\pm 4}$	$8_{\pm 10}$	$3_{\pm 2}$	$5_{\pm 4}$	$6_{\pm 9}$	$10_{\pm 8}$	$12_{\pm 11}$	$19_{\pm 10}$	$27_{\pm 11}$

on the small components of Elliptic and Rosenbrock's functions (*i.e.,* f_{21} and f_{25}) which can be the root cause of its bad performance.

The key difference between the spending patterns in Category 4 and 5 is that in the latter one the budget distributions are more biased towards the very large components (except in f_{23}) especially in CBCC2 and CBCC3 cases. For example, CBCC2 allocates up to 75% of the resources on $f_{25,10}$.

Figure 95 summarizes the unequal dimension size effects on CBCCs' efficiency. Similar to the previous figures, each column of plots corresponds to one basis function (from b_1 to b_5) while each row represents a category of problems (*i.e.,* Category 1, 4, and 5). The upward trends in the second and third rows show that CBCCs generally tend to spend more on the larger subproblems than the smaller ones. As mentioned earlier, Ackley's functions (*i.e.,* f_{18} and f_{23} that depicted in the third column) are exceptions.

Comparing the slope of trends in the middle row with the bottom row reveals that the significant differences in the component sizes in Category 5 result in a more nonuniform budget distribution. This confirms that CBCCs can adapt to the problems' degree of imbalance. In some cases such as CBCC2, however, the aggressive budget allocation may adversely result in a quality degrade rather than any improvement.

One can simply compare the behavior of CBCC variants based on the budget distribution provided in Figure 95. In general, CBCC1 either distributes the budget uniformly across all components or, at most, spends double on the most contributing components while other subproblems receive a virtually equal amount of resources. CBCC2, in contrast, may invest too much on the largest subproblem and almost nothing on the others. As usual, CBCC3 chooses a relatively moderate approach. In comparison with CBCC1, it spends more on the larger subproblems while in contrast with CBCC2 it secures a minimum of resources for even the smallest component in the set. This can be the key success

factor of CBCC3 in comparison with the other studied algorithms.

Figure 96 illustrates the portion of budget each component is received in each single trial of the CBCCs. In general, the figure confirms the finding from Figure 95 while is adds another perspective to the previous figure: the magnitude of the variance from one trial to another. Figure 96 suggests that the volatility in the received budget generally increases as the size of subproblems grows. Once again, Ackley's instances are exceptions. Another observation comes from comparing the second and the third rows of the figure. It seems the severe imbalance in Category 5 results in a large variance in the budget allocation patterns.

The difference between CBCCs' distribution that is reflected in Figure 96 is probably the most interesting observation from these plots. The figure hints that CBCC1 demonstrates very small variance in its spending in comparison with the other counterparts. The results from CBCC2, however, resembles a bipolar or multimodal distribution. It usually picks only one component to invest all the budget on it. The problem is that there is a high chance of selecting different components in different trials. Therefore, in some cases (*e.g.,* f_{16} and f_{20}) we observed that almost all components are selected as the most contributing component (and consumed all the resources) in at least one of the trials. Interestingly, the number of components that can be selected by CBCC2 decreases as the imbalance degree increases (*e.g.,* compare f_{16} and f_{20} with f_{21} and f_{25}). Finally, while the results obtained from CBCC3 show more volatility than of the CBCC1, its confidence in choosing the most contributing component is higher than CBCC2.

Category 6: Problems with Hybrid Landscapes

Table 917 compares the performance of examined algorithms on problems with mixed search landscapes. The table suggests that CBCC3 performs significantly better than the CC and other CBCC variants. Although CBCC3 shows to be the best performer virtually in all scenarios studied so far, its superiority is more evident here than ever. Surprisingly, CBCC1 and CBCC2 could only improve CC results on one of the problems (*i.e.,* f_{30}). On the rest, they appear to function worse than a simple round-robin budget allocation strategy. As expected, CBCC2 is the worst algorithm among the others.

To investigate the surprisingly unsatisfactory results of CBCC1 and CBCC2, we need to look at the allotted budget to each component. This is what Table 918 presents. According to the table, CBCC1 and CBCC2 agree on the most contributing component. Although the flexibility of CBCC2 allows it to distribute resources more aggressively (which usually result in very uneven distributions), the general pattern is very similar to what CBCC1 does. Conversely, in all cases but f_{30}, CBCC3 distributes the budget very differently. Indeed, it identifies some other component as the most contributing ones.

What is missing in Table 918 is that the association between the component index (k) and the basis functions is not immediately visible. Therefore, we provide another summary table, Table 919, that aggregates the share of budget spent on each basis function rather than every single component. Recall that each problem in this category consists of ten subproblems while only five unique basis functions are used. As a result, we aggregate (via summation) the portion of resources spent on subproblems with similar landscapes to produce Table 919.

It is clear from Table 919 that CBCC1 and CBCC2 always select b_4 subproblems as long as a component with such basis function exists. In this case, CBCC1 invests 12%–16% of the budget into solving these subproblems while CBCC2 may devote up to 28% of the resources on same components. The rest of the subproblems may uniformly receive around 9% or 7% –8% of the resources by CBCC1 and CBCC2, respectively.

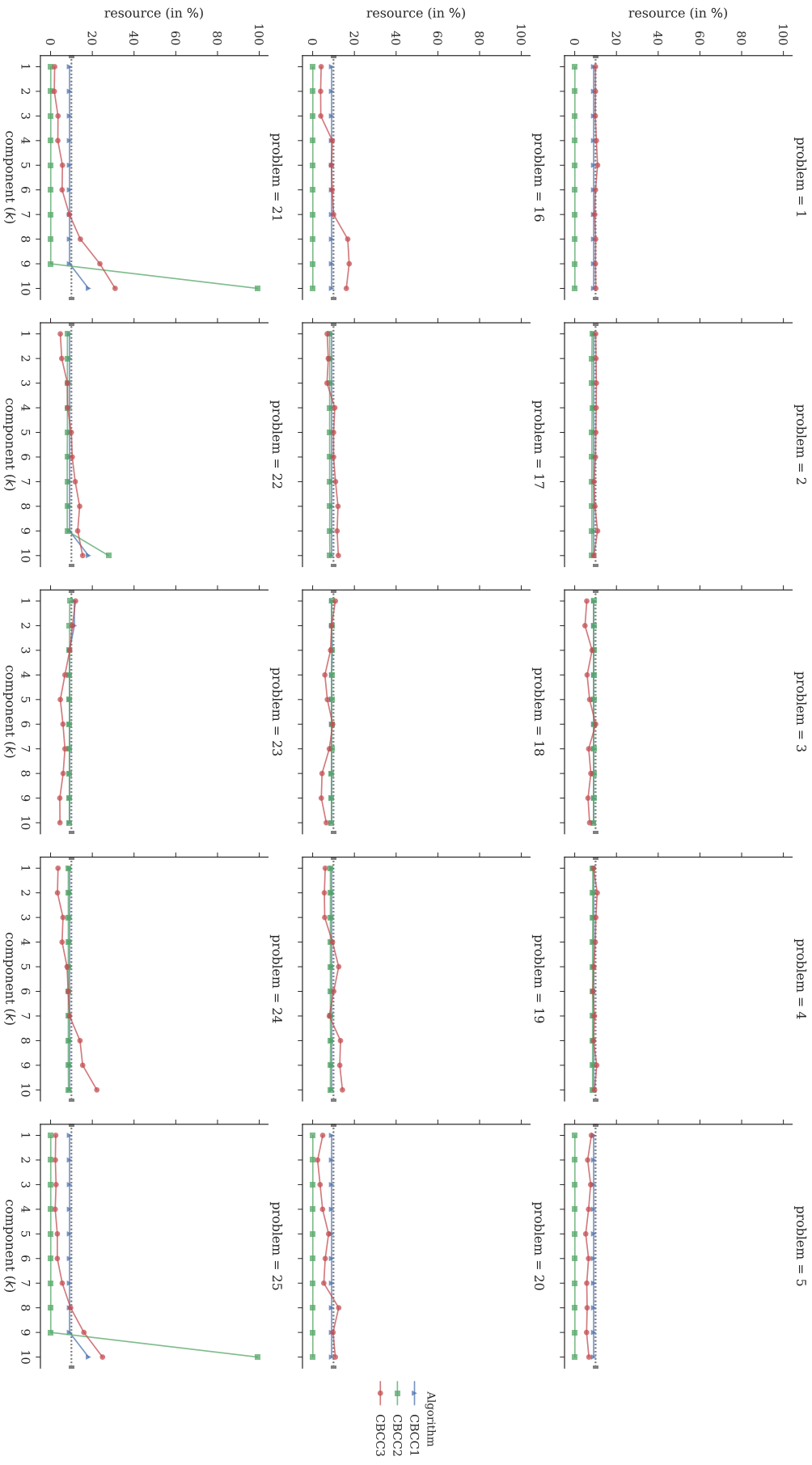


Figure 95: The median allocated budget to components of Category 1, 4, and 5. The black dash line at 10% indicates the uniform resource distribution.

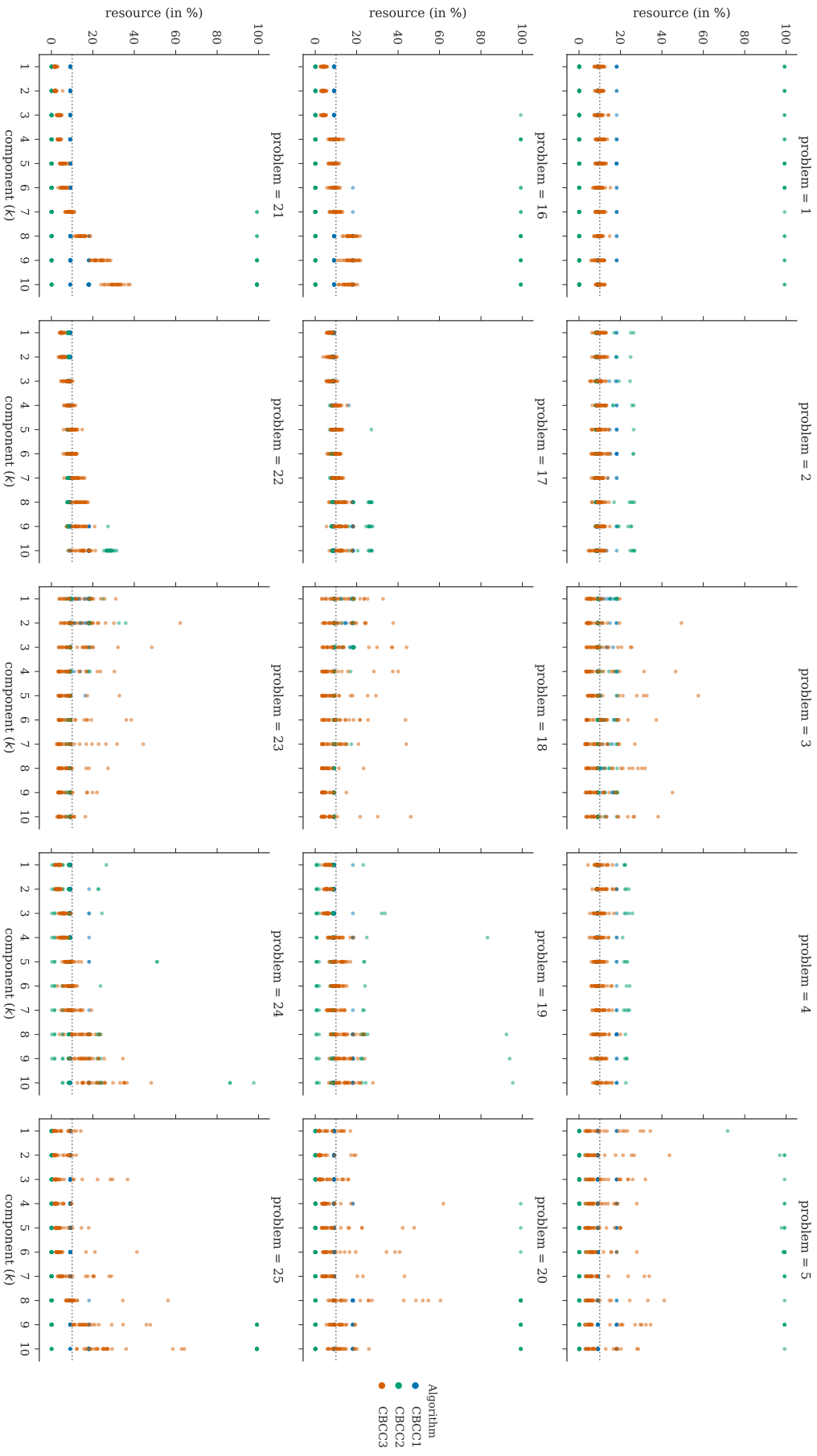


Figure 96: The allocated budget to components of Category 1, 4, and 5. The black dash line at 10% indicates the uniform resource distribution.

Table 917: Preliminary results of benchmarking a round-robin CC and three CBCCs on Category 6 (Hybrid Landscapes). For each problem, smallest median values are written in **boldface** font to indicate the best performing algorithm.

		CC	CBCC1	CBCC2	CBCC3
f_{26}	Median	6.47E+06	6.08E+06	7.65E+06	**2.18E+06**
	Mean	6.39E+06	6.36E+06	9.42E+06	2.18E+06
	Std	1.08E+06	1.10E+06	6.22E+06	4.35E+05
f_{27}	Median	5.59E+06	6.51E+06	4.06E+10	**1.09E+06**
	Mean	5.87E+06	6.42E+06	3.79E+10	1.08E+06
	Std	1.15E+06	1.05E+06	1.57E+10	2.18E+05
f_{28}	Median	6.06E+06	6.74E+06	7.39E+06	**2.00E+06**
	Mean	6.48E+06	6.91E+06	1.40E+07	2.05E+06
	Std	1.49E+06	1.40E+06	1.21E+07	5.17E+05
f_{29}	Median	6.39E+06	6.68E+06	7.54E+06	**2.22E+06**
	Mean	6.49E+06	6.91E+06	1.31E+07	2.21E+06
	Std	1.19E+06	1.68E+06	1.05E+07	3.89E+05
f_{30}	Median	8.43E+05	7.34E+05	7.05E+05	**5.12E+05**
	Mean	8.04E+05	7.60E+05	7.36E+05	4.82E+05
	Std	3.43E+05	2.26E+05	2.03E+05	2.36E+05

Table 918: The portion (in percentage) of allocated resources to solving each subproblem of Category 6 (Hybrid Landscapes) problems by CBCCs. Numbers are in $mean \pm_{std}$ format.

	CBCC	k									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
f_{26}	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$14_{\pm 5}$	$13_{\pm 6}$
	2	$8_{\pm 2}$	$8_{\pm 2}$	$8_{\pm 2}$	$8_{\pm 2}$	$8_{\pm 2}$	$8_{\pm 2}$	$8_{\pm 2}$	$8_{\pm 2}$	$19_{\pm 15}$	$17_{\pm 16}$
	3	$1_{\pm 0}$	$1_{\pm 0}$	$42_{\pm 6}$	$44_{\pm 5}$	$1_{\pm 0}$	$1_{\pm 0}$	$1_{\pm 0}$	$1_{\pm 0}$	$5_{\pm 3}$	$4_{\pm 4}$
f_{27}	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$14_{\pm 5}$	$13_{\pm 6}$
	2	$0_{\pm 1}$	$0_{\pm 1}$	$0_{\pm 1}$	$0_{\pm 1}$	$0_{\pm 1}$	$0_{\pm 1}$	$0_{\pm 1}$	$0_{\pm 1}$	$58_{\pm 49}$	$39_{\pm 50}$
	3	$1_{\pm 0}$	$1_{\pm 0}$	$48_{\pm 7}$	$45_{\pm 6}$	$1_{\pm 0}$	$1_{\pm 0}$	$1_{\pm 0}$	$1_{\pm 0}$	$2_{\pm 1}$	$2_{\pm 2}$
f_{28}	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$13_{\pm 5}$	$14_{\pm 5}$	$9_{\pm 0}$	$9_{\pm 1}$
	2	$7_{\pm 3}$	$7_{\pm 3}$	$7_{\pm 3}$	$7_{\pm 3}$	$7_{\pm 3}$	$7_{\pm 3}$	$22_{\pm 25}$	$21_{\pm 20}$	$7_{\pm 3}$	$7_{\pm 4}$
	3	$1_{\pm 0}$	$1_{\pm 0}$	$41_{\pm 5}$	$42_{\pm 5}$	$1_{\pm 0}$	$1_{\pm 0}$	$5_{\pm 3}$	$5_{\pm 2}$	$2_{\pm 1}$	$2_{\pm 2}$
f_{29}	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$12_{\pm 4}$	$16_{\pm 4}$	$9_{\pm 0}$	$9_{\pm 1}$
	2	$7_{\pm 3}$	$7_{\pm 3}$	$7_{\pm 3}$	$7_{\pm 3}$	$7_{\pm 3}$	$7_{\pm 3}$	$27_{\pm 27}$	$17_{\pm 15}$	$7_{\pm 3}$	$7_{\pm 4}$
	3	$1_{\pm 0}$	$1_{\pm 0}$	$39_{\pm 5}$	$45_{\pm 6}$	$1_{\pm 0}$	$1_{\pm 0}$	$4_{\pm 2}$	$4_{\pm 1}$	$2_{\pm 0}$	$2_{\pm 1}$
f_{30}	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$13_{\pm 5}$	$14_{\pm 5}$	$9_{\pm 0}$	$9_{\pm 1}$
	2	$7_{\pm 3}$	$7_{\pm 3}$	$7_{\pm 3}$	$7_{\pm 3}$	$7_{\pm 3}$	$7_{\pm 3}$	$28_{\pm 27}$	$16_{\pm 17}$	$7_{\pm 3}$	$7_{\pm 4}$
	3	$3_{\pm 1}$	$3_{\pm 1}$	$3_{\pm 1}$	$4_{\pm 1}$	$2_{\pm 1}$	$2_{\pm 1}$	$36_{\pm 9}$	$33_{\pm 11}$	$6_{\pm 5}$	$7_{\pm 6}$

Table 919: The portion (in percentage) of allocated resources to solving each subproblem of Category 6 (Hybrid Landscapes) problems by CBCCs. Numbers are in $mean \pm std$ format.

	CBCC	b_0	b_1	b_2	b_3	b_4	b_5
f_{26}	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$
	2	$8_{\pm 2}$	$8_{\pm 2}$	$8_{\pm 2}$	$8_{\pm 2}$	$8_{\pm 2}$	$8_{\pm 2}$
	3	$1_{\pm 0}$	$1_{\pm 0}$	$42_{\pm 6}$	$44_{\pm 5}$	$1_{\pm 0}$	$1_{\pm 0}$
f_{27}	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$
	2	$8_{\pm 2}$	$8_{\pm 2}$	$8_{\pm 2}$	$8_{\pm 2}$	$8_{\pm 2}$	$8_{\pm 2}$
	3	$1_{\pm 0}$	$1_{\pm 0}$	$42_{\pm 6}$	$44_{\pm 5}$	$1_{\pm 0}$	$1_{\pm 0}$
f_{28}	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$
	2	$8_{\pm 2}$	$8_{\pm 2}$	$8_{\pm 2}$	$8_{\pm 2}$	$8_{\pm 2}$	$8_{\pm 2}$
	3	$1_{\pm 0}$	$1_{\pm 0}$	$42_{\pm 6}$	$44_{\pm 5}$	$1_{\pm 0}$	$1_{\pm 0}$
f_{29}	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$
	2	$8_{\pm 2}$	$8_{\pm 2}$	$8_{\pm 2}$	$8_{\pm 2}$	$8_{\pm 2}$	$8_{\pm 2}$
	3	$1_{\pm 0}$	$1_{\pm 0}$	$42_{\pm 6}$	$44_{\pm 5}$	$1_{\pm 0}$	$1_{\pm 0}$
f_{30}	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$
	2	$8_{\pm 2}$	$8_{\pm 2}$	$8_{\pm 2}$	$8_{\pm 2}$	$8_{\pm 2}$	$8_{\pm 2}$
	3	$1_{\pm 0}$	$1_{\pm 0}$	$42_{\pm 6}$	$44_{\pm 5}$	$1_{\pm 0}$	$1_{\pm 0}$

In the absence of b_4 (*e.g.,* f_{27}), CBCC1 and CBCC2 select b_5 subproblems as the most important components. In this case, CBCC1 spends 13%–14% of the budget on each of b_5 subproblems and around 9% on each of the remaining components which is in line with its behavior in the rest of heterogeneous problems. In contrast, CBCC2 allocate almost all the budget on the b_5 subproblems and virtually nothing on the other components.

Is it evident from Table 918 and 919 that CBCC3 invests the resources in a completely different way. When b_1 subproblems exist in the set, CBCC3 selects them as the most important components. In their absence, it identifies b_4 components as the most contributing ones. Therefore, f_{30} is the only problem that all CBCCs agree on the most influencing component. This finding is aligned with an observation from Table 917 that shows f_{30} is the only problem in this category that all CBCCs can improve round-robin CC.

Another bold difference between CBCC3 and its precedents is that it does not uniformly distribute the remaining of the budget on the rest of (*i.e.,* all but not the most contributing) components. For example, Table 919 hints that in the case of f_{26} CBCC3 collectively devotes around 85% of the budget to b_1 components, less than 10% on the two b_4 subproblems, and only 1% to each of the remaining components. It shows very similar behavior in dealing with f_{28} and f_{29} as well. The only noticeable difference between the CBCC3 budget allocation in these problems and f_{26} is that in f_{28} and f_{29} cases the remaining of the budget is not uniformly distributed among all other components (*i.e.,* all except b_1 and b_4). Indeed, in these cases, the two b_5 components stands at the third place (after b_1 and b_4) and received a slightly larger portion of the budget than the rest of subproblems. This observation does not contradict with the distribution of resources in f_{26} as no b_5 subproblem is included there.

A closer look at f_{27} confirms that ranking of basis functions made by CBCC3 is very consistent. In this particular case, b_5 moves from the third place to the second as b_4 (the basis function that CBCC3 tends to select as the second most important subproblem)

is not included here. In the case of f_{30}, b_4 jumps to the first rank since b_1 subproblems which are the most contributing component from the CBCC3 point of view are missing. In this particular scenario, b_4 components are followed by b_5 (as usual) while b_3 subproblems receive the least amount of resources. In summary, we can conclude that CBCC3 ranks the basis functions in this order: $b_1 \succ b_4 \succ b_5 \succ b_0 \sim b_2 \succ b_3$, where $b_i \succ b_j$ indicates CBCC3 selects b_i more often than b_j and $b_i \sim b_j$ means b_i and b_j are selected with similar frequency.

Category 7: Conforming Imbalance Sources

The numbers in Table 920 suggests that CBCCs' performance in dealing with problems of Category 7 is analogous to the findings from Table2 and Table5 (respectively correspond to Category 3 and 5). For example, all algorithms achieve similar results on the Ackley's function (f_{33}). In other cases, CBCC3 delivers the best outcomes on average, whereas CBCC1 closely follows it. As usual, the results of CBCC2 on Elliptic and Rosenbrock's functions (f_{31} and f_{35}) are the worst among the others.

Table 920: Preliminary results of benchmarking a round-robin CC and three CBCCs on Category 7 (Conforming Imbalance Sources). For each problem, smallest median values are written in **boldface** font to indicate the best performing algorithm.

		CC	CBCC1	CBCC2	CBCC3
f_{31}	Median	6.15E+10	4.30E+10	8.25E+12	**1.75E+10**
	Mean	6.24E+10	4.19E+10	8.57E+12	1.78E+10
	Std	8.91E+09	7.83E+09	2.06E+12	2.02E+09
f_{32}	Median	2.46E+06	2.19E+06	1.99E+06	**1.71E+06**
	Mean	2.31E+06	2.18E+06	1.92E+06	1.69E+06
	Std	4.10E+05	1.85E+05	2.42E+05	1.92E+05
f_{33}	Median	**4.38E+04**	**4.38E+04**	**4.38E+04**	**4.38E+04**
	Mean	4.38E+04	4.38E+04	4.38E+04	4.38E+04
	Std	2.73E+01	2.52E+01	2.37E+01	3.06E+01
f_{34}	Median	2.95E+09	2.90E+09	2.93E+09	**2.46E+09**
	Mean	2.99E+09	2.57E+09	1.07E+11	2.22E+09
	Std	1.31E+09	1.21E+09	5.19E+11	9.58E+08
f_{35}	Median	4.30E+06	1.95E+06	8.74E+13	**8.25E+05**
	Mean	6.02E+06	2.73E+06	8.99E+13	1.21E+06
	Std	7.84E+06	2.35E+06	1.41E+13	1.12E+06

The key difference between the outcomes of contribution-aware CCs on Category 7 with their performance on Category 3 and 5 of problems (the parents of Category 7) is that CBCC1 could significantly improve round-robin CC on Rosenbrock's function (*i.e.,* f_{35}). For instance, compare the last row of Table 920 with the last rows of Table 911 and 915. To find the root cause of this observation, we need to take a closer look at the budget spent on solving each individual subproblem by CBCC1.

Table 921 suggests that the CBCC1 resource distributions have not been changed from Table 912. Indeed, it spends double on the most contribution component and distributes the rest of the budget uniformly. This is slightly different from the distributions in Table 916 which CBCC1 tends to invest a bit more on the second most contributing

Table 921: The portion (in percentage) of allocated resources to solving each subproblem of Category 7 (Conforming Imbalance Sources) problems by CBCCs. Numbers are in $mean \pm_{std}$ format.

	CBCC	C_k 10^1	10^2	10^3	10^4	10^5	10^6	10^7	10^8	10^9	10^{10}
		D_k 25	25	50	50	75	75	100	150	200	250
f_{31}	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$18_{\pm 1}$
	2	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$4_{\pm 20}$	$95_{\pm 21}$
	3	$1_{\pm 0}$	$1_{\pm 0}$	$1_{\pm 0}$	$2_{\pm 0}$	$2_{\pm 0}$	$3_{\pm 0}$	$5_{\pm 1}$	$12_{\pm 2}$	$24_{\pm 4}$	$48_{\pm 5}$
f_{32}	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$18_{\pm 1}$
	2	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$8_{\pm 0}$	$29_{\pm 1}$
	3	$2_{\pm 0}$	$2_{\pm 0}$	$3_{\pm 0}$	$4_{\pm 1}$	$5_{\pm 1}$	$6_{\pm 1}$	$9_{\pm 2}$	$15_{\pm 3}$	$22_{\pm 5}$	$31_{\pm 6}$
f_{33}	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$12_{\pm 4}$	$16_{\pm 5}$
	2	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$11_{\pm 4}$	$16_{\pm 5}$
	3	$3_{\pm 0}$	$4_{\pm 0}$	$4_{\pm 1}$	$4_{\pm 0}$	$6_{\pm 6}$	$6_{\pm 6}$	$8_{\pm 6}$	$11_{\pm 6}$	$21_{\pm 15}$	$34_{\pm 16}$
f_{34}	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 2}$	$9_{\pm 0}$	$12_{\pm 4}$	$15_{\pm 5}$
	2	$8_{\pm 3}$	$8_{\pm 3}$	$8_{\pm 3}$	$8_{\pm 3}$	$8_{\pm 3}$	$8_{\pm 4}$	$9_{\pm 6}$	$9_{\pm 6}$	$16_{\pm 18}$	$20_{\pm 19}$
	3	$2_{\pm 1}$	$2_{\pm 1}$	$3_{\pm 1}$	$3_{\pm 1}$	$4_{\pm 1}$	$5_{\pm 2}$	$8_{\pm 3}$	$13_{\pm 5}$	$21_{\pm 9}$	$39_{\pm 10}$
f_{35}	**1**	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$18_{\pm 1}$
	2	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$99_{\pm 1}$
	3	$1_{\pm 0}$	$2_{\pm 1}$	$2_{\pm 2}$	$3_{\pm 2}$	$5_{\pm 7}$	$5_{\pm 6}$	$7_{\pm 5}$	$17_{\pm 13}$	$21_{\pm 13}$	$36_{\pm 14}$

component (*i.e.,* 9^{th} subproblem) than the others. The positive correlation between the components' coefficients and their sizes make the improvement of CBCC1 on Category 7 bolder than its achievements on Category 3 and 5.

The numbers associated with CBCC2 budget allocation results in Table 921 have nothing to surprise us. In the case of Elliptic and Rosenbrock's functions (*i.e.,* f_{31} and f_{35}), CBCC2 spends almost all the budget on the most contributing subproblem. This aggressive budget allotment remains virtually no resources for any other components to be optimized. As a result, the CBCC2 performs even worse on these particular problems when compared with the traditional round-robin strategy.

Table 921 indicates on average CBCC2 tends to spend more on the subproblems with larger indices which also have larger sizes and coefficients. Comparing these pattern with the distributions in Table 912 and 916, we can conclude that the nonuniformity in coefficients has a stronger effect on CBCC2 than the unequal dimensionality. This strong similarity is more evident in b_1, b_3, and b_5. In particular, Table 916 suggests that CBCC2 tends to not invest a lot in high-dimensional Ackley's subproblems (*i.e.,* f_{23}) while Table 921 reveals that very large coefficients override the preference of CBCC2 as it selects 250- and 200−dimensional Ackley's subproblem more often than the smaller components just because they are multiplied by 10^9 and 10^{10}, respectively. This pattern is comparable with behavior of CBCC2 on selecting the components with the largest coefficients in Table 912 where all subproblems have the same size.

Figure 97 compares the performance of CBCCs in dealing with three categories of problems. The first two rows of subfigures represent the problems with unequal coefficients and nonuniform sizes, respectively. The last row, however, illustrates Category 7 which

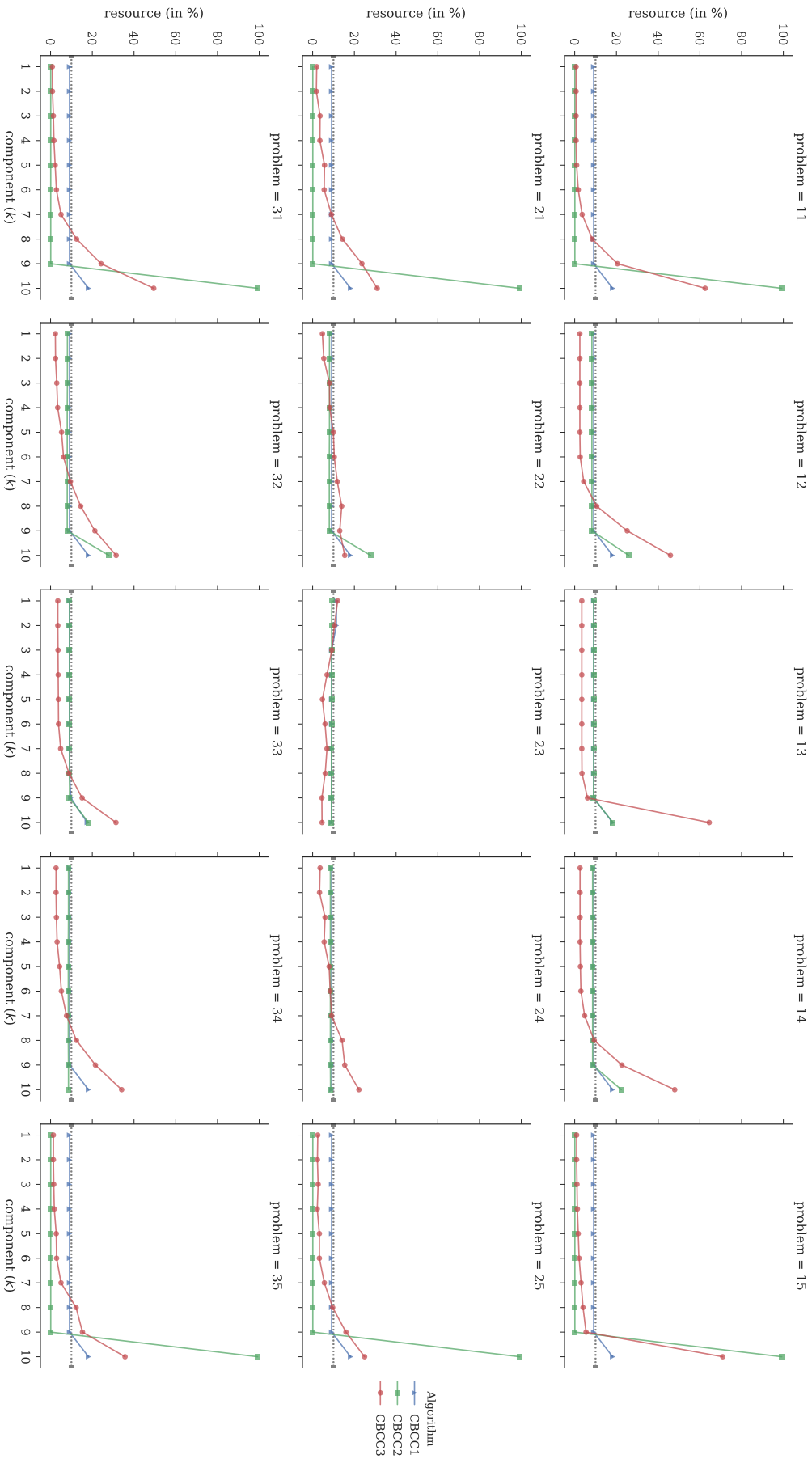


Figure 97: The median allocated budget to components of Category 3, 5, and 7. The black dash line at 10% indicates the uniform resource distribution.

is the combination of the other two. A quick comparison between the subplots from different rows reveals that the performance of CBCCs on Category 7 is closer to Category 3 than Category 5. This observation can be easily spotted specially in plots located in columns 2, 3, and 4 (*i.e.,* functions made based on b_2, b_3, and b_4 basis functions). This shows that the imbalanced in contribution imposed by nonuniform coefficients has a more significant effect than unequal components. Another observation is that the two sources of imbalance are conforming in practice as they magnify each others effect. The trends in depicted in the subfigures in the first and last columns (*i.e.,* b_1 and b_5) clearly show this phenomenon.

The scatter plots in Figure 98 confirm the observations from Figure 97. This figure also shows that the variances increase as the components' coefficients and sizes grow. Additionally, the figure suggests that the volatility in the budget allocations in Category 7 is slightly larger than Category 3.

Category 8: Confronting Imbalance Sources

Some measurements from Table 922 are aligned with what already observed in other cases studies. For example, CBCC2 and CBCC3 keep being the worst and best performers among the examined algorithms (particularly in b_1 and b_5 cases), respectively. However, there are some surprising facts as well. For instance, Table 922 reveals that CBCC1 cannot improve the round-robin CC results in most cases. In some cases, such as f_{36} and f_{39}, it functions worse than the baseline strategy. As another example, the improvements gained by CBCC3 over CC is not very significant either. These observations suggest that the negative correlation between components' sizes and their coefficients make it difficult for the CBCCs to determine the most contributing components. For a deeper analysis, we clearly need more detailed information of the distributions of the allocated resources.

Table 922: Preliminary results of benchmarking a round-robin CC and three CBCCs on Category 8 (Confronting Imbalance Sources). For each problem, smallest median values are written in **boldface** font to indicate the best performing algorithm.

		CC	CBCC1	CBCC2	CBCC3
f_{36}	Median	5.26E+08	5.64E+08	7.78E+11	**4.61E+08**
	Mean	5.93E+08	7.13E+08	7.41E+11	4.65E+08
	Std	3.73E+08	5.10E+08	1.88E+11	7.16E+07
f_{37}	Median	1.14E+05	1.24E+05	1.19E+05	**1.09E+05**
	Mean	1.17E+05	1.20E+05	1.19E+05	1.09E+05
	Std	9.29E+03	1.06E+04	9.74E+03	8.39E+03
f_{38}	Median	4.26E+04	4.25E+04	4.25E+04	**4.24E+04**
	Mean	4.22E+04	4.21E+04	4.25E+04	4.24E+04
	Std	2.12E+03	2.11E+03	1.80E+02	1.90E+02
f_{39}	Median	6.13E+07	7.16E+07	7.20E+07	**5.64E+07**
	Mean	6.30E+07	6.99E+07	9.86E+07	5.85E+07
	Std	9.42E+06	1.39E+07	8.08E+07	1.63E+07
f_{40}	Median	1.99E+05	1.35E+05	2.64E+06	**6.74E+04**
	Mean	2.55E+05	2.21E+05	1.87E+12	1.04E+05
	Std	2.08E+05	3.10E+05	2.49E+12	1.04E+05

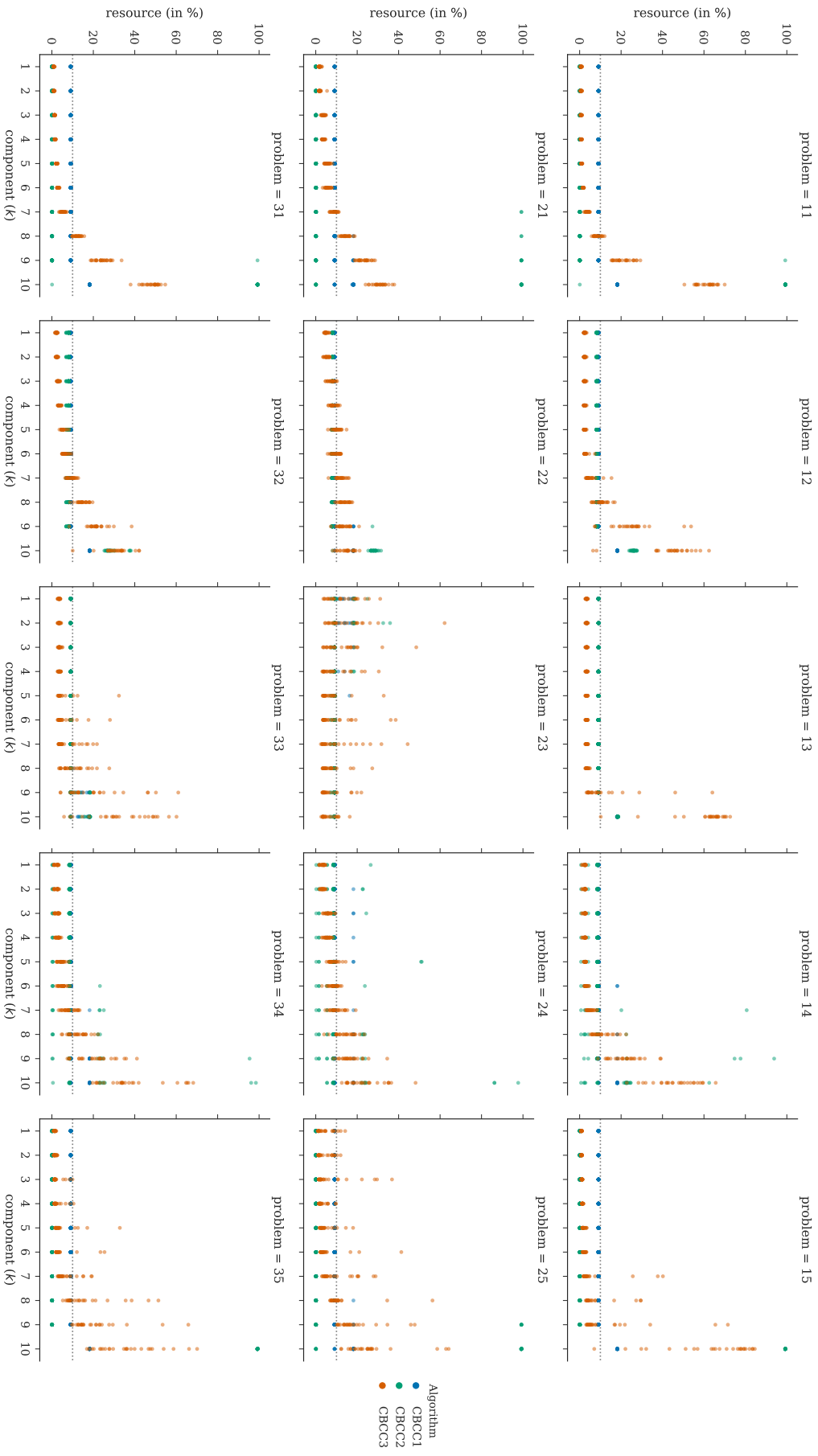


Figure 98: The allocated budget to components of Category 3, 5, and 7. The black dash line at 10% indicates the uniform resource distribution.

Table 923 provides the summary statistics we need for a rigorous analysis of CBCCs' budget spending on Category 8. As the table suggests, CBCC1 tends to select components with the larger coefficients more often than subproblems with higher dimension sizes. The only exception here is f_{39} (*i.e.,* b_4) which not only CBCC1 but all CBCCs favor larger subproblems over larger coefficients.

Table 923: The portion (in percentage) of allocated resources to solving each subproblem of Category 8 (Confronting Imbalance Sources) problems by CBCCs. Numbers are in $mean \pm_{std}$ format.

	CBCC	C_k									
		10^{10}	10^9	10^8	10^7	10^6	10^5	10^4	10^3	10^2	10^1
		D_k									
		25	25	50	50	75	75	100	150	200	250
f_{36}	**1**	$12_{\pm 4}$	$11_{\pm 3}$	$12_{\pm 4}$	$9_{\pm 2}$	$10_{\pm 3}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 1}$
	2	$28_{\pm 45}$	$20_{\pm 40}$	$44_{\pm 50}$	$8_{\pm 27}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 0}$	$0_{\pm 1}$
	3	$6_{\pm 2}$	$5_{\pm 1}$	$10_{\pm 2}$	$8_{\pm 2}$	$11_{\pm 2}$	$9_{\pm 2}$	$10_{\pm 1}$	$13_{\pm 1}$	$14_{\pm 2}$	$14_{\pm 3}$
f_{37}	**1**	$18_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 1}$
	2	$21_{\pm 2}$	$10_{\pm 2}$	$9_{\pm 2}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 1}$
	3	$12_{\pm 3}$	$10_{\pm 3}$	$14_{\pm 5}$	$13_{\pm 3}$	$12_{\pm 3}$	$9_{\pm 1}$	$9_{\pm 2}$	$8_{\pm 2}$	$6_{\pm 1}$	$5_{\pm 2}$
f_{38}	**1**	$17_{\pm 3}$	$11_{\pm 3}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 1}$
	2	$19_{\pm 8}$	$11_{\pm 7}$	$9_{\pm 1}$	$9_{\pm 1}$	$9_{\pm 1}$	$9_{\pm 1}$	$9_{\pm 1}$	$9_{\pm 1}$	$9_{\pm 1}$	$9_{\pm 2}$
	3	$29_{\pm 15}$	$21_{\pm 12}$	$18_{\pm 15}$	$9_{\pm 6}$	$6_{\pm 4}$	$4_{\pm 1}$	$4_{\pm 2}$	$3_{\pm 0}$	$3_{\pm 0}$	$3_{\pm 1}$
f_{39}	**1**	$9_{\pm 0}$	$9_{\pm 2}$	$10_{\pm 3}$	$9_{\pm 0}$	$9_{\pm 0}$	$11_{\pm 4}$	$9_{\pm 2}$	$11_{\pm 3}$	$11_{\pm 4}$	$12_{\pm 5}$
	2	$8_{\pm 6}$	$8_{\pm 5}$	$9_{\pm 7}$	$8_{\pm 5}$	$10_{\pm 11}$	$7_{\pm 3}$	$8_{\pm 5}$	$9_{\pm 7}$	$18_{\pm 26}$	$16_{\pm 27}$
	3	$6_{\pm 3}$	$6_{\pm 2}$	$12_{\pm 4}$	$11_{\pm 4}$	$12_{\pm 3}$	$10_{\pm 2}$	$11_{\pm 3}$	$10_{\pm 2}$	$11_{\pm 3}$	$10_{\pm 4}$
f_{40}	**1**	$12_{\pm 4}$	$11_{\pm 3}$	$14_{\pm 5}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 0}$	$9_{\pm 1}$
	2	$30_{\pm 28}$	$5_{\pm 9}$	$39_{\pm 46}$	$7_{\pm 19}$	$3_{\pm 3}$	$3_{\pm 3}$	$3_{\pm 3}$	$3_{\pm 3}$	$3_{\pm 3}$	$3_{\pm 4}$
	3	$10_{\pm 3}$	$10_{\pm 3}$	$10_{\pm 11}$	$4_{\pm 3}$	$9_{\pm 9}$	$6_{\pm 8}$	$11_{\pm 12}$	$12_{\pm 11}$	$12_{\pm 6}$	$16_{\pm 7}$

According to Table 923, the spending behavior of CBCC2 is comparable with the allocations made by CBCC1, but only more aggressive. For instance, CBCC2 may spend up to 44% of the budget (on average) on a subproblem with a large coefficient while several large components may receive almost no resources (*e.g.,* all components of f_{36} with dimension sizes larger than 50). As usual, this strong nonuniformity in the budget allocation in CBCC2 is more evident in b_1 and b_5 functions (see f_{36} and b_{40} in Table 923).

The most interesting observation in Table 923 belongs to performance of CBCC3. In the case of f_{36}, CBCC3 selects larger subproblems more often than any other components. In dealing with f_{37} and f_{38}, by contrast, it chooses subproblems with larger coefficients more than other components. In the case of f_{39}, however, the budget distribution is very different. In this particular scenario, CBCC3 prefers to spend most of the resources on the subproblems with middle sizes and medium coefficients. Therefore, the components with extreme conditions (*e.g.,* too large but small coefficients or small but having too large coefficients) receive fewer resources than the subproblems with moderate conditions.

The numbers in Table 923 hints that the performance of CBCC3 on the last problem is in the opposite of its behavior on the previous one. It indeed selects the subproblem with the extreme conditions more often than the subproblems with average sizes and

coefficients. While the allotted budgets on f_{39} and f_{40} seem opposite, the final objective values of the solution found by CBCC3 (reported in Table 922) suggest it can beat all other examined algorithms in both cases. Therefore, one can conclude that the optimum resource allocation schema not only depends on the components sizes and coefficients, but also on the features of search landscape. Overall, these study reveals new dimensions of component selection and budget allocation strategies that we could never identify using the previously proposed benchmarks.

A deep investigation in Table 923 reveals a special pattern in the first four columns. The table suggests that in general CBCCs assign more resources to the third component than the second and forth components. The main reason behind this trend is that the first and second components have the same dimension sizes, but differ in coefficients. Therefore, the second components receive significantly less resources than the first ones. Similarly, the third and the forth components have similar sizes while the coefficient of the third ones is 10 times larger than the forth component. As a result, the forth subproblems are selected less frequently than the third ones. Putting these two phenomena together results to an increase in the third components' budget in comparison with the second and the forth subproblems.

Figure 99 compares Category 8 with 5 and 7. All problems in these categories suffer from unequal component sizes. The only difference between the problems in these categories is their components' coefficients. As mentioned earlier, all coefficients in Category 5 are equal (*i.e.,* $C_k = 1$), whereas these values in Category 7 and 8 are increasing or decreasing functions of component indices, respectively.

A quick look at Figure 99 reveals that the first two rows share more similarities than the last row. Indeed, the strong dissimilarity between the trends in the third row and the rest of plots in the figure confirm that both nonuniform coefficients and unequal subproblem sizes have a significant effect on the selection patterns of CBCCs. In general, we can see these factors virtually cancel each others' effect such that in most cases the resources are distributed uniformly. Note that this type of phenomenon cannot be studied using the previously proposed benchmark sets.

A deeper investigation in Figure 910 unveils an important effect of mixed imbalance sources. As the last row of the figure suggests, CBCC1 and CBCC2 demonstrate a multimodal budget distribution pattern. It means, they severely struggle to identify the most contributing component such that in different trials they allocate the budgets very differently. It seems to have subproblems with similar size but unequal coefficients (compare first and third components with the second and the forth, respectively) makes the budget assignment even a harder task. This observation is more visible in f_{36}, f_{37}, and f_{40}.

9.5 Chapter Summary

We begin this chapter by reviewing the previously used test sets to benchmark the budget allocation mechanism of CC (see Section 9.1). Then in Section 9.2, we specified the important features that a comprehensive problem set should ideally have to be able to study different aspects of contribution imbalance problem. We used these features as a guideline to develop a more comprehensive test set in Section 9.3 where we systematically designed a set of 40 large-scale problems with imbalanced contributions as the most extensive testbed in assessing the resource distribution mechanism of Contribution-Aware CCs. The set includes eight categories of problems each of them reflects a unique scenario with respect to the imbalance type and level. Following a systematic design, the categories

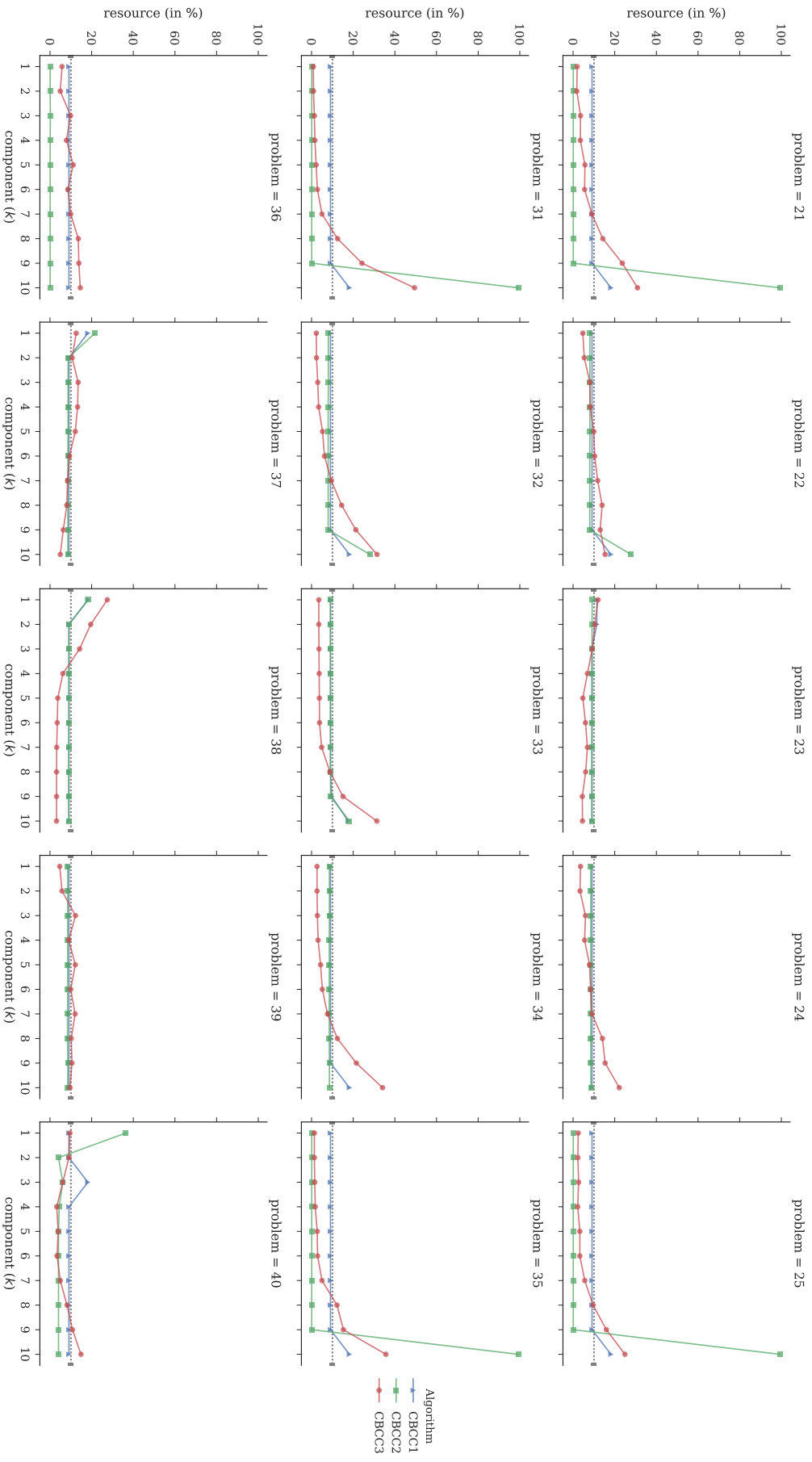


Figure 99: The median allocated budget to components of Category 5, 7, and 8. The black dash line at 10% indicates the uniform resource distribution.

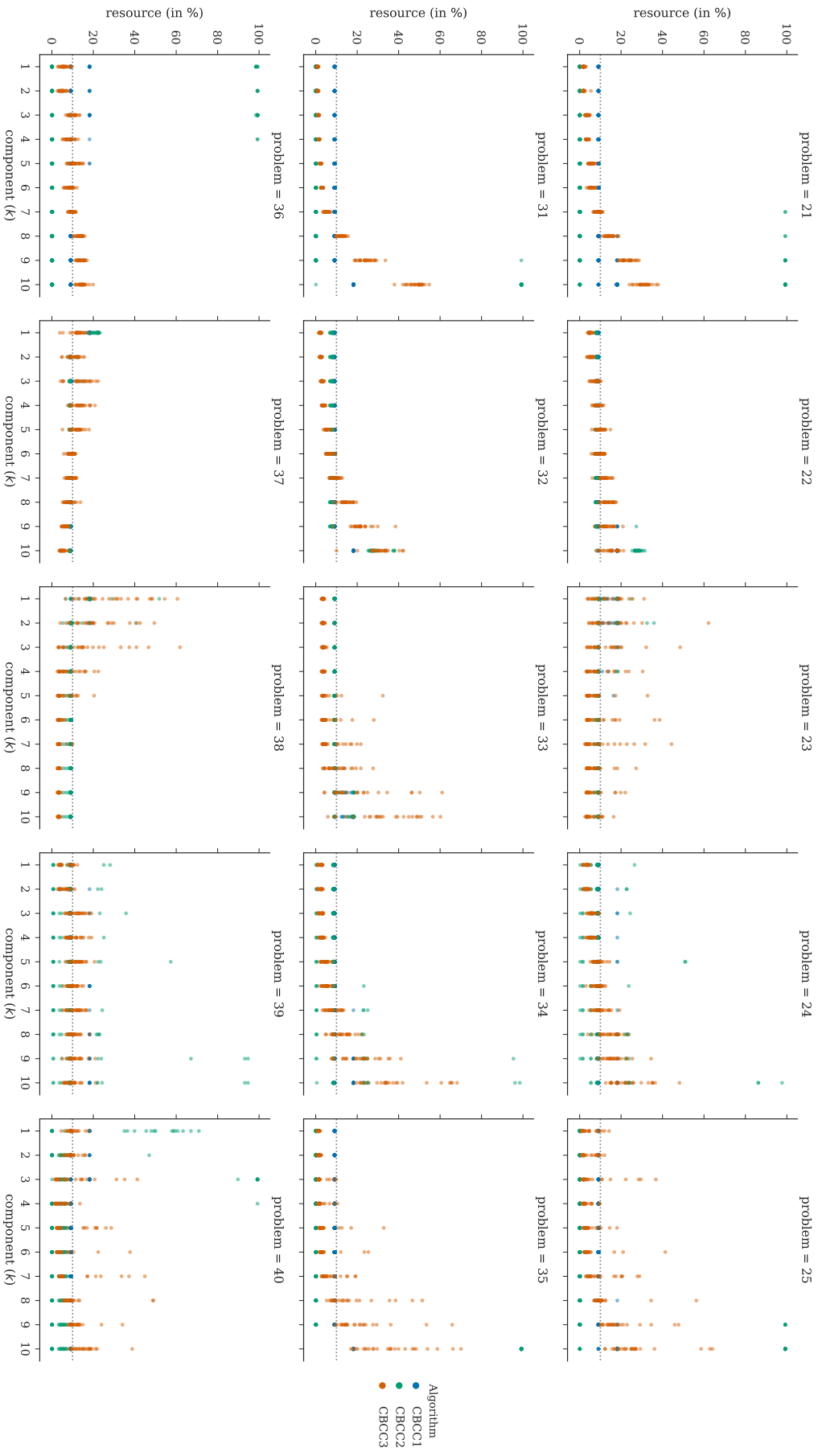


Figure 910: The allocated budget to components of Category 5, 7, and 8. The black dash line at 10% indicates the uniform resource distribution.

can be easily compared to investigate the effect of each source or level of contribution imbalance.

Our numerical simulation that presented in Section 9.4 confirmed that the proposed testbed provide several important insights about the studied algorithms that were not possible to gain using the previous benchmarking sets. As a result, the proposed functions provide a unique opportunity to investigate the performance of budget allocation strategies in CC algorithms as well as decomposition techniques and optimization algorithms.

In this chapter we only had room to show the potentials of the testbed by benchmarking a round-robin CC and three variants CBCCs. We expand this line of research by also benchmarking other budget allocation strategies in the next chapter using the proposed set.

CHAPTER 10

Bandit-Based Cooperative Coevolution

In the previous chapters, we showed that Contribution-Aware Cooperative Coevolutionary (CACC) algorithms can achieve a significant improvement over the traditional round-robin strategy on a range of problems with imbalanced components. However, they still suffer from some limitations such as having a large number of influencing parameters and *ad hoc* heuristics with no theoretical support.

To address these shortcomings, we extend the CC framework to a new generic framework capable of learning the contribution of each component using Multi-Armed Bandit techniques. This approach results in a more economical use of the limited computational resources than the previously proposed CACCs. Furthermore, the proposed framework leverages the findings from the vast amount of researches that have been done on bandit techniques and their applications in online budget allocation.

Towards the end of this chapter, we study different aspects of the proposed framework in the light of extensive numerical experiments. Our empirical results confirm that even a simple bandit-based credit assignment scheme can significantly improve the performance of CCs on large-scale continuous problems, leading to competitive performance as compared to the state-of-the-art algorithms. Finally, we show that the existing contribution-aware CCs can be modeled as special cases of the proposed framework.

10.1 Introduction

As mentioned in the previous chapters, the round-robin resource allocation mechanism of standard CCs is not necessarily the best strategy when the component of the given objective function exhibit some sort of contribution imbalance. In Chapter 7 we reviewed several techniques that track the performance of subfunctions to estimate their contribution towards solving the main function. As opposed to the traditional round-robin budget assignment, these techniques aim to distribute the computational resources based on the estimated contribution of each subfunction. Our sensitivity analysis on a class of contribution-aware CCs (CACCs) called Contribution-Based CCs (CBCC) shows that they can make a better use of limited resources even in the presence of decomposition error (see Chapter 8).

Experimental results in Chapter 9 confirm that CACCs show a promising performances on numerous problems. However, efficient resource allocation in imbalance black-box optimization problems is still a challenging task. A number of factors contributes to the complexity of the task are:

1. The potential contributions of the components are usually unknown to users and optimizers, especially in the case of black-box problems [Omidvar et al. 2015]. Therefore, one cannot effectively preallocate the budget before carefully tracking the performance of the optimizer on each and every subfunctions.

2. In many cases, the distributions of contributions are not linearly correlated with the amounts of resources that should be allocated to the components. For example, having a component with twice as large as the other one's contribution (in an arbitrary scale) does not necessarily mean that it should receive a portion of resources two times larger than the other component's budget. Therefore, CC algorithms must be equipped with an effective resource management mechanism (as a mapping from contribution space to budget space).

3. A competent mechanism is needed to maintain a right balance between the budget spent on the contribution learning (*a.k.a.* exploration phase) and further optimization of the most contributing component found so far (*a.k.a.* exploitation phase). Because of the uncertainty in the estimated contributions and also the fact that their distributions may change dramatically over time, these estimations should be updated frequently. For example, a component that is identified to be contributing significantly the beginning stage of the search may contribute less or close to none at a later stage. Furthermore, the contribution learning costs valuable resources that could be spent on the optimization of the most rewarding components. Therefore, preserving the balance between contribution leaning and component optimization is vital in such an uncertain and dynamic environment.

In spite of the improvements that CACCs demonstrate over the classic CC framework, they still suffer from some shortcomings:

1. A number of CACCs adopted some *ad hoc* heuristics to learn the contributions and allocate the resources (*e.g.,* the CBCC family [Omidvar et al. 2011; 2016]). As a result, they face difficulties to adapt to the dynamics of the search process. Furthermore, the effect of their parameters are difficult to validate and their values hard to tune.

2. Some of the proposed CACCs techniques are often combinations of several other algorithms (*e.g.,* MOFBVE [Mahdavi et al. 2016c; 2017]). This extra complexity brings lots of parameters into the framework such that the effects and contributions of each part to the overall outcome are difficult to identify.

3. The performance of the available CACCs is only examined on a limited number of test functions. For a better statistical analysis, they need to be assessed by a comprehensive set of large-scale imbalanced problems which covers a variety of cases (similar to the benchmarking set that we proposed in Chapter 9).

In this chapter, we model the resource allocation problem of CC framework as a *dynamic multi-armed bandit* problem [Kuleshov and Precup 2014]. We propose a modular framework to deal with the imbalanced contribution issue in the context of large-scale black-box optimization. Adopting the theoretically-sound multi-armed bandit solvers into this architecture, which we refer to as Bandit-Based Cooperative Coevolution (BBCC), has at least two merits:

1. We can leverage the extensive literature on bandit problems and solvers to evaluate the proposed framework and better understand its behavior. As a result, the theoretical and practical significance of BBCC will be much easier to appreciate than those previous *ad hoc* techniques.

2. The modularity of BBCC will provide users the flexibility to choose any appropriate decomposition strategies, reward functions, credit assignment formulations, bandit solvers, and optimization methods to plug into the framework. This feature will further facilitate sensible comparative studies on the topic of resource allocation.

The rest of the chapter is organized as follows: In Section 10.2, we provide a very brief introduction to multi-armed bandit techniques and their applications in the online resource allocation. Then, Section 10.3 introduces the proposed bandit-based framework and a few of its instances. Afterward, Section 10.4 provides a range of case studies, sensitivity analyses and comparative studies to demonstrate the capacities of BBCC. Finally, Section 10.5 concludes the chapter and highlights some potential topics for future work.

10.2 An Introduction to Multi-Armed Bandits

In this chapter, we model the computational budget allocation of the CC framework as a *dynamic credit assignment* problem. In a typical credit assignment problem, two or more actions with unequal benefits are competing against each other. The objective of the solver is to find a sequence of actions that maximizes the accumulated rewards in a given time horizon.

A large number of credit assignment problems have been addressed by Multi-Armed Bandit (MAB) techniques [Gittins et al. 2011]. A heuristic MAB solver samples the reward distributions by exploring the action space to estimate the expected long-term profit from the execution of each action. At the same time, the solver should exploit the knowledge and fire the most rewarding action as often as possible to achieve the maximum accumulated reward. Therefore, maintaining a balance between exploration (*i.e.,* examining different actions) and exploitation (*i.e.,* taking the best action repeatedly) in the action space is absolutely vital. Preserving this balance is especially critical in non-stationary problems where the distributions of the rewards change over time [Li et al. 2014, Fialho et al. 2010a; 2009].

MAB techniques have been extensively used in the metaheuristics for adaptive operator selection [Li et al. 2014, Consoli et al. 2014, Fialho et al. 2010b], DE strategy selection [Fialho et al. 2010b, Gong et al. 2011], and component size adaptation in the CC framework [Omidvar et al. 2014b]. In all of these applications, the MAB algorithm consists of four essential modules:

Action Set: The action set is the collection of all available options which remain unchanged during the optimization. The set of evolutionary operators in operator selection [Fialho et al. 2010b] and the predefined component sizes in adaptive variable grouping in CC [Omidvar et al. 2014b] are two examples of action sets.

Reward Function: The reward function measures the immediate benefit of taking a particular action. For example, the difference between the fitness values of parents and their offspring can be used to measure the goodness of the chosen evolutionary operator in an adaptive operator selection problem. Due to the natural stochasticity of metaheuristics, the reward values may significantly change over time. In many cases, not only the reward values but also the statistical distributions that these

values are sampled from may dramatically change during optimization process. For instance, the reward gained by firing a particular operator may significantly drop when the population is stagnated or converged.

Credit Function: The credit function translates the immediate rewards to long-term utility of taking an action. In the simplest form, credit is just accumulated rewards coming form selecting an action over time. Ideally, the accumulation feature devised in credit function should minimize the volatility forced by the reward dynamics. The credit (*a.k.a.* utility) of an action estimates its usefulness if it will be executed repeatedly until the optimization process terminates.

Budgeting Strategy: The budgeting strategy is a mapping function from credit space back into action space. In other words, strategy selects the next action based on the estimated credits of executing that action for the rest of trials. A strategy can be as simple as executing the action with the maximum credit estimation. However, since strategy should maintain a balance between the exploration and exploitation in the action space, it usually has to set a part of remaining budget aside for further explorations or updating the credit estimation.

In the following sections, we explain how the resource allocation problem in CC can be formulated as a MAB problem, and how this formulation is implemented in our Bandit-Based CC (BBCC) framework.

10.3 Bandit-Based Cooperative Convolution Framework

Analogous to MAB agents that we briefly reviewed in Section 10.2, the BBCC framework consists of four fundamental modules:

Component Pool: The component pool is the collection of subfunctions that is generated by the decomposition algorithm which resembles the action set in MAB literature. A BBCC instance has to select one component at a time from this pool for further optimization.

Improvement Measure: This measure evaluates the immediate changes in the objective value of the main function after optimizing a particular component for a number of iterations (*i.e.,* one epoch). The improvement measure in BBCC is very similar to reward function in the MAB taxonomy.

Contribution Estimator: This estimator predicts the contribution of optimizing a component to the improvement of the quality of the final solution. Similar to credit functions in MAB literature, contribution estimator can be as simple as the average of the past measured improvement or as complex as a regression model that predicts unforeseen improvements of future actions.

Component Selector: This is very similar to MAB strategies as it selects the next component to be optimized based on the estimated contributions. Again, the component selector should maintain a working balance between component exploration and exploitation during the search process.

The high-level idea of BBCC is to firstly form a component pool of separate groups of decision variables, then explore and measure the improvements gained by optimizing each of these components for a limited number of iterations. These values help BBCC

Algorithm 7 The Bandit-Base Cooperative Coevolutionary Framework

```
1:  function BBCC(f, D, N, , δ_t)
2:      (X, F[:, 0]) ← initialization(f, N, D)          ▷ Population initialization
3:      x⃗ ← initContextVector(X, F)                     ▷ Initial Context Vector
4:      D ← initComponentPool(f)                        ▷ Equation (10.1)
5:      K ← size(D)                                     ▷ Number of components
6:      μ[:, 0] ← initContribution(K, ∞)                ▷ Initial contribution
7:      t ← 0                                           ▷ Epoch counter
8:      repeat
9:          t ← t + 1                                   ▷ Increase epoch counter
10:         k ← componentSelector(μ)                    ▷ Equation (10.18)
11:         X[:, D[k]] ← optimize(f, X[:, D[k]], x⃗, δ_t)  ▷ One epoch optimization
12:         F[:, t] ← evaluate(f, X)                    ▷ Re-evaluate
13:         x⃗ ← updateContextVector(X, F, x⃗)            ▷ Update Context Vector
14:         δ[k, t] ← improvementMeasure(F)             ▷ Equation (10.3)
15:         μ[k, t] ← contributionEstimator(δ)          ▷ Equation (10.7)
16:         D ← updateComponentPool(f, D)               ▷ Equation (10.2)
17:     until termination() = True                      ▷ Termination
18:     return X[arg min(F[:, t]), :]                   ▷ Return final solution
```

to estimate the long-term contribution of each component to the overall fitness value. Finally, the most contributing component is selected and optimized based on the estimated contributions. Since contributions change over time, the component pool exploration should be carried out several times in the course of optimization to update the estimations. As a result, the component selector may change the budgeting schema from time to time according to the observed improvements and the dynamics of contribution estimations.

A simplified pseudocode of BBCC framework is presented in Algorithm 7. As lines 2–3 state, BBCC starts with initializing the population and building an initial context vector. Then, the component pool $\mathcal{D}$ is formed (see Subsection 10.3.1) and its size is determined at lines 4 and 5. The objective of BBCC is to select and optimize one of these K components at a time such that the overall fitness improvement is maximized.

In line 6, BBCC initiates the matrix $\boldsymbol{\mu}$ which stores the estimated contributions. When there is no prior knowledge about the contributions of the components, it is recommended to initialize $\boldsymbol{\mu}$ with an effectively large value to ensure all components are examined at least once (∞ at line 6 indicates a large positive number).

The main loop at lines 8–17 starts by selecting a subfunction using a component selection algorithm (see Subsection 10.3.4). The selected component is then optimized using an arbitrary optimizer for one epoch or δ_t iterations. As a result of this step, the F and $\vec{\mathbf{x}}$ may need to be updated. The main challenge here is to maintain a good balance between exploring all components to improve and update the estimations and optimizing the most contributing component as many times as possible.

In line 14, BBCC assesses the effectiveness of the optimization of the recently selected component using an improvement measure (see Subsection 10.3.2). Next, BBCC updates the contribution estimations based on the recent measured improvement (*i.e.,* $\boldsymbol{\delta}$) and other factors, *e.g.,* the number of times the component is optimized so far or the magnitude of the remaining budget (see Subsection 10.3.3) in line 15. This historical information is then recorded to be used by the component selector at the next round.

All steps inside the loop are repeated until a termination criterion is met (line 17). At

this point, the solution with the highest fitness value (or minimum objective value in minimization tasks) is returned as the ultimate solution (line 18). For the sake of preserving the simplicity, the control parameters *e.g.,* δ_f in Equation (10.6) or ϵ in Equation (10.19) are excluded from Algorithm 7. In the followings, we discuss the BBCC modules in more details.

10.3.1 Component Pool

The component pool is conceptually similar to the action set in the MAB taxonomy. Assume a D-dimensional optimization problem is decomposed into K distinct subfunctions, either manually or automatically:

$$\mathcal{D} = g(f) \tag{10.1}$$

where the component pool $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_K\}$ is essentially the superset of all subfunctions of function f that are formed by an arbitrary decomposition technique g. Here, $\mathcal{D}_k$ indicates the indices of the decision variables that form the k^{th} component. Notably, any decomposition technique can be acquired in Equation (10.1) to initiate the component pool. For example, if a dynamic decomposition technique is adopted, the component pool should be updated every few iterations:

$$\mathcal{D}^{(t)} = g(f, \mathcal{D}^{(t-1)}) \tag{10.2}$$

In Algorithm 7, the equations 10.2 and 10.1 are denoted as `initComponentPool` and `updateComponentPool`.

10.3.2 Improvement Measure

In the absence of *a priori* knowledge about component contributions, BBCC needs a tool to measure the goodness of each action. This metric is related to the concept of immediate or practical reward in the MAB literature. In our particular problem, the goodness of optimizing a component depends on its contribution to improving the objective value of the final solution. Therefore, we define the improvement measure as a function of the fitness fluctuations when a component is being optimized. In the most general form, we define

$$\delta_k^{(t)} = \Delta(\mathbf{F}^{(t)}), \tag{10.3}$$

where $\mathbf{F}^{(t)}$ is an $N \times t$ matrix that contains the complete history of the fitness values of all candidate solutions till epoch t. Note that in Equation (10.3), we implicitly assume that k^{th} component is optimized at epoch $t-1$. Since only one component is optimized at a time, the $\delta_i^{(t)}$ for all $i \neq k$ is zero.

Usually, we do not need to record all the fitness values during the optimization to measure the improvement of a component. For example, we can assess the gained improvement of minimizing the k^{th} component at epoch t only based on the fitness value of the best solutions before and after the optimization of that component:

$$\delta_k^{(t)} = \overset{*}{f}^{(t-1)} - \overset{*}{f}^{(t)}, \tag{10.4}$$

where $\overset{*}{f}^{(t-1)}$ and $\overset{*}{f}^{(t)}$ are the objective values of the best solution before and after optimizing the k^{th} component for one epoch [1].

Generally speaking, the magnitudes of fitness values vary significantly as optimization progresses. For example, some δ_k may take larger values at the early epochs and smaller

[1] In maximization tasks, the order of the terms in the above definition should be reversed.

values towards the end. In such cases, using the simple difference may result in over-emphasizing the earlier trials rather than the recent observed improvements. To lessen this effect, one can simply *normalize* the fitness improvement as:

$$\delta_k^{(t)} = \frac{\overset{*}{f}^{(t-1)} - \overset{*}{f}^{(t)}}{\overset{*}{f}^{(t-1)} + \varepsilon}, \tag{10.5}$$

where ε is a small constant that is added to avoid division by zero. Assuming $\overset{*}{f}^{(t)}$ is always positive, Equation (10.5) guarantees that $\forall t \, \forall k, \; 0 \leq \delta_k(t) < 1$.

The *quantization* is another approach to deal with the significant variations in objective values during optimization. For example, it is possible to binarize the fitness improvement as:

$$\delta_k^{(t)} = \begin{cases} 1 & \text{if } \overset{*}{f}^{(t-1)} - \overset{*}{f}^{(t)} > \delta_f \\ 0 & \text{otherwise} \end{cases}, \tag{10.6}$$

where δ_f is a tuning parameter that can take absolute (*e.g.,* 10^{-8}) or relative (*e.g.,* $\overset{*}{f}^{(t)}/100$) values to control the sensitivity of the measurement on the marginal improvements.

The generality of the BBCC framework allows us to adopt any of the aforementioned improvement measures, as well as many others (see `improvementMeasure` in Algorithm 7). Practitioners can choose the most effective form of improvement measure according to properties such as the sensitivity to small improvements (*e.g.,* Equation (10.4)) or robustness in noisy environments (*e.g.,* Equation (10.6)).

10.3.3 Contribution Estimator

The contribution estimation in the BBCC framework is similar to the credit (*a.k.a.* value and utility) function in Reinforcement Learning [Sutton and Barto 1998]. The objective of this module is to translate the measured improvements (as immediate rewards) to estimated contributions (as long-term utility). Therefore, the general form of the contribution estimator is:

$$\mu_k^{(t)} = M(\boldsymbol{\delta}^{(t)}), \tag{10.7}$$

where $\boldsymbol{\delta}^{(t)}$ denotes the measured improvements of all components till epoch t.

In general, this task can be done in three ways: *contribution value* estimation, *contribution rank* estimation, or a combination of both. The first approach generalizes the past measured improvements to forecast the contribution values in the next optimization epochs. These projected values will be used to prioritize the optimization of the subfunctions. In contrast to the value estimation, the rank-based approach directly sorts the components according to their past performance without any explicit calculation of contribution magnitudes. The hybrid techniques, as their name suggests, are combinations of the other two categories. In what follows, we discuss these approaches in further details and provide some examples from the application of similar MAB approaches in adaptive metaheuristics.

A: Contribution Value Estimation

In a stationary situation, the expected contribution of the k^{th} component can be approximated as the mean of the past fitness improvements:

$$\mu_k^{(t)} = \frac{1}{n_k^{(t)}} \sum_{i=1}^{t} \delta_k^{(i)} = \frac{1}{|\boldsymbol{\lambda}_k^{(t)}|} \sum_{i \in \boldsymbol{\lambda}_k^{(t)}} \delta_k^{(i)}, \tag{10.8}$$

where $\boldsymbol{\lambda}_k^{(t)}$ denotes a set of length $n_k^{(t)}$ that contains all epoch indices that the k^{th} component has been optimized until epoch t. As mentioned earlier, for all the other epochs (*i.e.,* $i \notin \boldsymbol{\lambda}_k^{(t)}$) the improvements are assumed to be zero, and hence, the contributions remain unchanged. Whenever this component is selected to be optimized again, the new epoch number should be added to $\boldsymbol{\lambda}_k^{(t)}$ and $n_k^{(t)}$ should be increased by one unit. Based on Central Limit Theorem, as t tends toward infinity, $\mu_k^{(t)}$ in Equation (10.8) approaches its true value if all δ_k are drown from a fixed statistical distribution.

Since the distribution of δ_k may change multiple times during optimization process, the stationary assumption in Equation (10.8) becomes invalid. For example, it is very common for optimizers to be trapped into a local optimum or stagnated during the search process. In any of these scenarios, the magnitude of the recent δ_k drops dramatically. Therefore, Equation (10.8) becomes inefficient as it takes a long time to adapt to the dynamics, especially when such event happens after many epochs (*i.e.,* large $n_k^{(t)}$ values).

This issue can be addressed in two ways: *passive* or *active*. In the passive approach, we assume that the improvement distributions are stationary at least in a bounded time window. By holding this assumption, we can substitute $\boldsymbol{\lambda}_k^{(t)}$ by $\overset{L}{\boldsymbol{\lambda}}_k^{(t)} = \langle \lambda_1, \ldots, \lambda_L \rangle$ which only contains the epoch indices of the last L non-zero improvements of k^{th} component prior to epoch t. Therefore, Equation (10.8) should be substitute by:

$$\mu_k^{(t)} = \frac{1}{L} \sum_{l=1}^{L} \delta_k^{(\lambda_l)}. \tag{10.9}$$

The window length L in Equation (10.9) can be kept constant or tuned adaptively. To have a smoother sliding window, we could adopt a weighted averaging method:

$$\mu_k^{(t)} = \frac{1}{L} \sum_{l=1}^{L} W_i \cdot \delta_k^{(\lambda_l)}, \tag{10.10}$$

where $\forall l, W_l \in [0, 1]$ and $\sum_{l=1}^{L} W_l = 1$. Usually larger weights are used for the more recent observations. A similar approach is to adopt a forgetting factor which does not need to store all previous δ_k values (only having $\delta_k^{(t)}$ is enough);

$$\mu_k^{(t)} = \alpha \cdot \delta_k^{(t)} + (1 - \alpha) \cdot \mu_k^{(t-1)}, \tag{10.11}$$

where $\alpha \in (0, 1]$ controls the length of the memory horizon such that larger values for α will result in a shorter memory.

In the active approach, we assume that the improvement distributions are stationary unless we find an evidence that proves otherwise. In other words, an active estimator calculates contributions based on all observed improvements (similar to Equation (10.8)), while at the same time compares the current statistics of the improvement with the previous records. Once a significant change has been detected, we clear all records and build the model from scratch (*e.g.,* reinitializing $\boldsymbol{\delta}$ and $\boldsymbol{\mu}$). This approach is very helpful when a dynamic decomposition algorithm is adopted [Rainville et al. 2013]. In the metaheuristic literature, some statistical change detection tests such as Page-Hinkley have been used in similar situations [Fialho et al. 2010b].

B: Rank-Based Estimations

There are multiple ways to rank components based on their previous improvements. Let $\overset{L}{\sigma}{}_k^{(t)}$ denote the sum of the last L improvements gained by optimizing the k^{th} component:

$$\overset{L}{\sigma}{}_k^{(t)} = \sum_{i=1}^{L} \delta_k^{(\lambda_i)}, \tag{10.12}$$

and $\gamma_k^{(t)}$ is its position after we sort all components based on the values of $\overset{L}{\boldsymbol{\sigma}}{}^{(t)}$ in a descending order. Now, the score of the k^{th} component can be defined as:

$$\mu_k^{(t)} = \frac{\alpha^{\gamma_k^{(t)}}}{\sum_{i=1}^{K} \alpha^{\gamma_i^{(t)}}}, \tag{10.13}$$

where $\alpha \in (0, 1)$ is a decay factor. Equation (10.13) resembles the famous *Normalized Exponential Function* which have been widely used in statistical analysis and machine learning techniques to represent categorical distributions. Since the calculated scores μ_k are in the range $(0, 1)$ that add up to 1, they can be interpreted as the probabilities of contributing rather than the contribution values.

Another possible rank-based definition that can be adopted is the well-known *sum-of-ranks* that works as follows [Li et al. 2014, Fialho et al. 2010b]. Let the last L improvements of all components be sorted in descending order and $\overset{L}{\boldsymbol{\gamma}}{}^{(t)} = \langle \gamma_1, \ldots, \gamma_L \rangle$ contains the latest L ranks. Then, the sum-of-ranks score is defined as:

$$\mu_k^{(t)} = \frac{\sum_{i=1}^{L} w_{k,i} \cdot \alpha^{\gamma_i}}{\sum_{j=1}^{K} \sum_{i=1}^{L} w_{j,i} \cdot \alpha^{\gamma_i}}, \tag{10.14}$$

$$\text{where } w_{k,i} = \begin{cases} L - \gamma_i & \text{if } \gamma_i \text{ is associated with } k^{\text{th}} \text{ component} \\ 0 & \text{otherwise} \end{cases} \tag{10.15}$$

Equation (10.15), guarantees that components with better ranks (or smaller γ_k) are multiplied by larger coefficients. Note that there are two pronounced differences between Equation (10.13) and Equation (10.14): i) In the former equation, we only consider the most recent ranking of the components, whereas in the latter a weighted sum of the latest L ranks is used. ii) Equation (10.13) calculates $\mu_k^{(t)}$ based on the latest L ranks of the k^{th} component whereas in Equation (10.14) we only look at the latest L ranks of all components.

C: Hybrid Estimators

By combining the aforementioned approaches, one can have the best of two worlds. The main disadvantage of contribution value estimation is that it can overestimate the contribution of some components if the improvement range is too wide (see Gini's index of CCAOI in Subsection 7.4.1). On the other hand, a very small difference in recorded improvement of two or more components has the potential to change their orders and dramatically affect the performance of rank-based techniques. Therefore, incorporating the magnitude of the observed improvements into the rank-based estimator can help us to sort the components while minimizing the risk of over/underestimations.

The simplest hybridization is to define a function based on the improvement values and plug it into the definition of a rank-based estimator. For example, one can modify Equation (10.13) to:

$$\mu_k^{(t)} = \frac{\overset{L}{\sigma}{}_k^{(t)} \cdot \alpha^{\gamma_k^{(t)}}}{\sum_{i=1}^{K} \overset{L}{\sigma}{}_i^{(t)} \cdot \alpha^{\gamma_i^{(t)}}}, \tag{10.16}$$

where $\overset{L}{\sigma}{}_k^{(t)}$ is the sum of the last L improvements of k^{th} component (defined in Equation (10.12)) and $\gamma_k^{(t)}$ is its rank among all components (used in Equation (10.13)). It is worth noting that any of the aforementioned contribution estimation methods can be adopted in BBCC (see `contributionEstimator` in Algorithm 7). In addition, it is possible to make a combination of them to maintain better accuracy in very noisy or dynamic situations.

10.3.4 Component Selector

The main goal of the component selector, which is conceptually close to the budgeting strategy module of an MAB algorithm, is to select a component to be optimized at the next epoch based on the estimated contributions. Note that the selection algorithm should maintain a good balance between component pool exploitation (*i.e.,* allocating more resources to the most contributing component) and exploration (*i.e.,* trying different components to improve/update the estimations).

Several techniques have been developed in the MAB literature for sustaining the exploration-exploitation balance. With no exception, any of these algorithms can be adopted in BBCC as a component selector. We can define a stochastic selector (*e.g., semi-uniform* and *probability-based* selectors) as:

$$p_k^{(t+1)} = S(\boldsymbol{\mu}^{(t)}), \tag{10.17}$$

where $p_k^{(t+1)}$ is the probability of the k^{th} component being selected for the next epoch and $\boldsymbol{\mu}^{(t)}$ denotes the estimated contributions of all components at epoch t. The deterministic selectors (*e.g., interval-based* selectors) can be defined as:

$$k^{(t+1)} = S(\boldsymbol{\mu}^{(t)}), \tag{10.18}$$

where $k^{(t+1)}$ denotes the component that is selected to be optimized in the next epoch. This step is denotes as `componentSelector` at line 10 of Algorithm 7. In the following, we briefly review some representative approaches from each category.

A: Semi-uniform Component Selectors

In the exploration phase, semi-uniform techniques uniformly allocate a portion of the budget to all components regardless of their contributions. In the exploitation phase, these algorithms spend a part of the remaining resources to the component with the highest estimated contribution. ϵ-greedy is a famous example of semi-uniform techniques:

$$k^{(t+1)} = \begin{cases} \overset{*}{k} & \text{if } \texttt{rand}(1) \geq \epsilon \\ \lceil \texttt{rand}(1) \cdot K \rceil & \text{otherwise} \end{cases}, \tag{10.19}$$

where $\overset{*}{k}{}^{(t)}$ denotes the component with the highest estimated contribution at epoch t, $\texttt{rand}(1)$ generates a random number in range $[0, 1]$ and $\lceil . \rceil$ denotes the ceiling function.

In Equation (10.19), the $\epsilon \in (0, 1)$ can be either constant or variable. As the above equation states, decreasing ϵ will result in a more greedy selection. Some of other variants of semi-uniform techniques *e.g.,* ϵ-first, GreedyMix and LeastTaken are discussed in [Vermorel and Mohri 2005].

B: Probability-based Component Selectors

To avoid the uniform budget distribution over all non-optimal components that happens in semi-uniform selectors, the probability-based strategies map each $\mu_k^{(t)}$ to a probability of being selected. These probabilities are usually proportional to the expected contributions. SoftMax (or Boltzmann Exploration) is a widely-used technique from this category [Omidvar et al. 2014b]. In SoftMax, the probability of selecting the k^{th} component in the next epoch is:

$$p_k^{(t+1)} = \frac{e^{\frac{\mu_k^{(t)}}{\tau}}}{\sum_{i=1}^{K} e^{\frac{\mu_i^{(t)}}{\tau}}}, \tag{10.20}$$

where τ is a temperature parameter that controls exploration-exploitation balance. For $\tau = 0$, Softmax only selects the most contributing component, while it picks all components uniformly when τ tends towards infinity.

Some probability-based techniques *e.g.,* Probability Matching and Adaptive Pursuit algorithms explicitly guarantee the minimum and maximum likelihoods that a component can be selected [Kim et al. 2012]. Other forms of these techniques, *e.g.,* Adaptive Probability Matching, Power Probability Matching, Exp3 and SoftMix are explained in [Vermorel and Mohri 2005, Kim et al. 2012].

C: Interval-based Component Selectors

These algorithms compute the confidence interval which indicates to what extent we are confident of the accuracy of estimated contributions. The well-known Upper Confidence Bound (UCB) algorithms [Consoli et al. 2014] follow the rule of *'optimism in the face of uncertainty'* [Fialho et al. 2010b]. This means the components that are less explored so far have a higher chance of being selected even if their expected contributions are not very high. For example, UCB1 selects the next component according to the following rule:

$$k^{(t+1)} = \underset{1 \leq i \leq K}{\arg\max} \left(\mu_i^{(t)} + \alpha \cdot \sqrt{\frac{2 \ln(t)}{n_i^{(t)}}} \right), \tag{10.21}$$

where α is a scaling factor that controls the trade-off between exploration and exploitation. As the equation suggests, by decreasing α the algorithm greedily selects the most contributing component more often than the less-explored components that might have some undiscovered potentials. The Price Of Knowledge and Estimated Reward (POKER) is another example of interval-based strategies which explicitly takes the horizon (*i.e.,* the maximum number of objective function calls) into account. POKER and other variants of UCB are described in [Kuleshov and Precup 2014, Vermorel and Mohri 2005].

There exist extensive literature about the advantages and disadvantages of these techniques in different applications [Vermorel and Mohri 2005, Kim et al. 2012, Kuleshov and Precup 2014]. However, there is no single study that compares them in the context of computational budget assignment when dealing with imbalanced functions. We postpone such deep analysis on the sensitivity of BBCC to the adopted component selector to future work.

10.3.5 Notes and Discussions

Before proceeding to the empirical studies, there are a few special topics worth mentioning.

A: BBCC and Other Contribution-Aware Techniques

As mentioned earlier, BBCC is a general framework rather than a single specific technique. This means, one can easily create special instances of the BBCC framework by adopting different techniques for each of its modules. Indeed, it can be shown that all available CACCs are special cases of the BBCC framework.

For example, the normalized improvement in Equation (10.5) is used as the improvement measure in CBCC1 and CBCC2. Conversely, CCFR uses the absolute difference between the fitness values of the best solutions before and after epoch t as the improvement measure (*i.e.*, $|\hat{f}^{(t-1)} - \hat{f}^{(t)}|$).

The contribution estimator of CBCC1 and CBCC2 is simply the average of improvement over all epochs as presented in Equation (10.8). CBCC3, however, only considers the last recorded improvement which is equivalent to Equation (10.9) with $L = 1$. In CCFR, contributions are calculated according to Equation (10.11) where $\alpha = 0.5$. It also uses a stagnation detection technique to reset the estimations if it finds a stagnant subpopulation. In fact, this is an active approach to deal with the dynamics in the contributions that was discussed in Subsection 10.3.3.

CCFR and all variants of CBCC adopt the same semi-uniform component selection strategy. In their exploitation phase, they always select the most contributing component: $k^{(t+1)} = \arg\max \mu_i^{(t)}$. In contrast, all components are optimized in a round-robin fashion in the exploration phase: $k^{(t+1)} = (k^{(t)} \mod K) + 1$. The main difference between these algorithms is the rules they determine to switch between exploration and exploitation phases (see Chapter 7).

B: Dynamic Grouping/Decomposition

As clearly stated in Algorithm 7, it is possible to adopt a dynamic decomposition technique (*e.g.*, [Omidvar et al. 2014b]) in the BBCC framework. In this case, BBCC should update $\boldsymbol{\delta}$ and $\boldsymbol{\mu}$ matrices whenever the dimension indices of a component (*i.e.*, $\mathcal{D}_k$) have been changed, a component is removed, or a new component is created [Rainville et al. 2013]. It is also possible to inherent these values from parent(s) to the child component(s) if the new components are created by splitting or merging operators. Obviously, too frequent changes in the decomposition adversely affect the learning capability of CACCs, including BBCC instances, as the recorded information becomes inoperative very soon.

C: Epoch Length

Although all CC techniques should tune the epoch length (*i.e.*, δ_t value) to achieve the maximum performance, this parameter could have more influence on the CACC techniques, including BBCC instances. Choosing a large value for δ_t has advantages such as improving the robustness of the measured improvements in the face of fitness fluctuations as well as decreasing the overhead costs of component switching. On the other hand, a too large δ_t may increase the delay in response to the dynamics of measured improvements. Since the number of function evaluations is fixed, as the epoch length increases, the number of epochs must be decreased. This would decrease the exploration opportunities of the algorithms. In Subsection 10.4.3 we will show that for some implementations of BBCC,

there are a range of parameter settings for which the performance of the algorithm is less affected by δ_t.

10.4 Experiments, Results and Discussions

This section consists of five major parts:

1. Case studies that investigate the behavior of an instance of BBCC called BBCC1 (see 10.4.1) in different scenarios,
2. Sensitivity analysis on BBCC1 generic parameters,
3. Comparison studies between BBCC1 and a number of round-robin CCs,
4. Comparison studies between BBCC1 and other CACCs, and
5. Comparison studies between BBCC1 and a number of state-of-the-art algorithms for large-scale problems.

We use the CEC'13 LSGO benchmark functions to compare BBCC with other available techniques. This test suite is the most recent and widely used large-scale optimization benchmark in this domain. However, only 8 of 15 functions are partially separable with some degrees of imbalanced contributions. In addition, the nonuniform contributions introduced in these functions are set arbitrarily which does not cover all common cases. Therefore, we use 30 large-scale partially separable imbalanced functions from Chapter 9 to carry out the case studies and sensitivity analysis. Since these problems were not available when the previous algorithms were proposed, it is not possible to compare them against the other researchers' results. Instead, we use the widely studied CEC'13 LSGO benchmarks to compare BBCC instances with other methods.

10.4.1 Experiments Setup

In all experiments, we chose to study the simplest possible implementation of BBCC (see Subsection 10.4.1). This helps us to avoid unnecessary complications and minimize the effects of the factors that are out of the scope of this study. This series of experiments shows that even a simple BBCC implementation can provide very competitive results. Thus, the main objective of the following experimental studies is merely to provide a proof of concept and show the potentials of this framework.

BBCC1: The Simplest BBCC Implementation

Algorithm 8 shows the pseudocode of BBCC1. A normalized fitness improvement from Equation (10.5) with epoch length $\delta_t = 50$ is used to measure the improvements after optimizing each component. As in Equation (10.8), the basic average of all improvements is adopted for contribution estimation. We adopt ϵ-greedy from Equation (10.19) as the simplest component selector. The ϵ value is set to 0.1 which means the most rewarding component will be selected with at least 90% chance. The remaining 10% of the budget will be spent uniformly on all components regardless of their estimated contribution. In BBCC1, we adopt the ideal grouping as the decomposition algorithm and DE/rand/1/bin as the optimizer.

The population size $N = 50$, weighting factor $F = 0.5$, and crossover rate $CR = 0.9$ are used for the experiments (see Table 101).

Table 101: Parameter values that are used in the numerical experiments.

Parameter	Value	Description
N	50	Population Size
N_e	$3.0e{+}6$	Maximum Function Evaluation
δ_t	50	Epoch Length
F	0.5	DE Scaling Factor
CR	0.9	DE Crossover Rate

Algorithm 8 An instance of Bandit-Based Cooperative Coevolutionary Framework

function BBCC1($f, D, N, \mathbf{l}, \mathbf{u}, \delta_t$)
  $t \leftarrow 0$ ▷ Epoch counter
  $\mathbf{X} \leftarrow \texttt{rand}(N, D) \circ (\mathbf{u} - \mathbf{l}) + \mathbf{l}$ ▷ Initialization
  $\mathbf{F}[:, t] \leftarrow \texttt{evaluate}(f, \mathbf{X})$ ▷ Evaluation
  $\overset{*}{i}_t \leftarrow \texttt{argmin}(\mathbf{F}[:, t])$ ▷ Find best solution
  $\overset{*}{f}_t \leftarrow \mathbf{F}[\overset{*}{i}, t]$ ▷ Best objective value
  $\mathbf{v} \leftarrow \mathbf{X}[\overset{*}{i}, :]$ ▷ Initial Context Vector
  $\mathcal{D} \leftarrow \texttt{idealGrouping}(f)$ ▷ Decomposition Equation (10.1)
  $K \leftarrow \texttt{size}(\mathcal{D})$ ▷ Number of components
  $\mathbf{n}[1{:}K] \leftarrow 0$
  $\boldsymbol{\mu}[1{:}K, t] \leftarrow \infty$ ▷ Initial contribution
  repeat
    $t \leftarrow t + 1$ ▷ Increase epoch counter
    if $\texttt{rand}(1) \geq \epsilon$ **then**
      $k \leftarrow \texttt{argmax}(\boldsymbol{\mu}[:, t])$ ▷ Exploitation Equation (10.19)
    else
      $k \leftarrow \texttt{ceil}(\texttt{rand}(1) \cdot K)$ ▷ Exploration Equation (10.19)
    $\mathbf{n}[k] \leftarrow \mathbf{n}[k] + 1$
    $\mathbf{X}[:, \mathcal{D}[k]] \leftarrow \texttt{DE}(f, \mathbf{X}[:, \mathcal{D}[k]], \mathbf{v}, \delta_t)$ ▷ One epoch DE
    $\mathbf{F}[:, t] \leftarrow f(\mathbf{X})$ ▷ Re-evaluate
    $\overset{*}{i}_t \leftarrow \texttt{argmin}(\mathbf{F}[:, t])$ ▷ Find best solution
    $\mathbf{v}[\mathcal{D}[k]] \leftarrow \mathbf{X}[\overset{*}{i}, \mathcal{D}[k]]$ ▷ Update Context Vector
    $\overset{*}{f}_{t-1} \leftarrow \overset{*}{f}_t$ ▷ Old best objective value
    $\overset{*}{f}_t \leftarrow \mathbf{F}[\overset{*}{i}, t]$ ▷ New best objective value
    $\boldsymbol{\delta}[k, t] \leftarrow (\overset{*}{f}_{t-1} - \overset{*}{f}_t)/(\overset{*}{f}_{t-1} + 10^{-8})$ ▷ Equation (10.5)
    $\boldsymbol{\mu}[k, t] \leftarrow \texttt{sum}(\delta[k, :])/\mathbf{n}[k]$ ▷ Equation (10.8)
  until $t \cdot N \cdot \delta_t \geq 3000 \cdot D$ ▷ Termination
  return $\mathbf{X}[\texttt{argmin}(\mathbf{F}[:, t]), :]$ ▷ Return final solution

Performance Metrics

In this study, we consider two aspects when analyzing the performance of the algorithms: components *selection distribution* and final *solution quality*. These numbers are recorded for 51 independent runs of each algorithm where the maximum number of objective function evaluations for each run is fixed to $3e{+}06$.

For the case studies, we analyze the selection distributions to study the performance of BBCC1 in identifying the most contributing component and effective budget allocation. These distributions are calculated based on the number of times each component is selected during the optimization. These statistics, which are presented using a variety of diagrams, indicate how precisely BBCC1 can find the most contributing component, and how efficient the exploration-exploitation balance is maintained.

We assess the final solution quality in terms of the objective value of the best solution found in each single run. We use this metric especially for sensitivity analysis and comparative studies. In addition to the fitness mean and standard deviation of the final solutions, *nWins* scores, *Win-Tie-Loss* values and Friedman rankings are also reported in the comparison tables [García et al. 2010].

Each nWins value indicates the number of times BBCC1 significantly improves other algorithms, subtracting the number of times it performs significantly worse than its competitors. Therefore, nWins scores take values in the range $\{-N_a, +N_a\}$ when BBCC1 is compared with N_a other algorithms. The values in each Win-Tie-Loss triple respectively indicates the number of times BBCC1 performs significantly better than, statistically similar to, or significantly worse than a particular competitor. For both metrics, the non-parametric Mann-Whitney U test (*a.k.a.* Wilcoxon rank-sum test) with 95% confidence interval is adopted as the significance test. Any comparison with a p-value less than 0.05 is considered as statistically similar performances. In cases where the detailed results of an algorithm were not available, we substitute Mann-Whitney Utest with the standard two-sided t-test. Although we cannot guarantee that the result distributions satisfy the t-test assumptions, at least it allows us to perform some significant test in the absence of detailed results.

For further statistical analyses, we also perform Friedman and Quade statistical significance tests [García et al. 2010]. For each comparison study, we provide Friedman ranks and orders, as well as Friedman and Quade p-values. When the overall p-values are smaller than 0.05, which signals a significant difference between the compared algorithms, we perform post-hoc tests to find the pairs that are responsible for that large difference. In this case, we adjust p-values using the famous Holm's and Bonferroni's correction techniques to control family-wise error rate [García et al. 2010].

10.4.2 Case Studies

In this part, we conduct a series of experiments to study the performance of BBCC1 component selection mechanism (see Algorithm 8). In particular, we analyze the effectiveness of component selector in four different scenarios: i) uniform contributions, ii) unequal component coefficients, iii) nonuniform component sizes, and iv) problems with heterogeneous subfunctions from Chapter 9. We deliberately choose these scenarios to investigate the effect of each imbalance type and level in a controlled environment.

The Uniform Contributions

In the first part of the case studies, we investigate the behavior of BBCC1 on balanced problems. For this case, we use f_1–f_5 where each of them has 10 components with equal

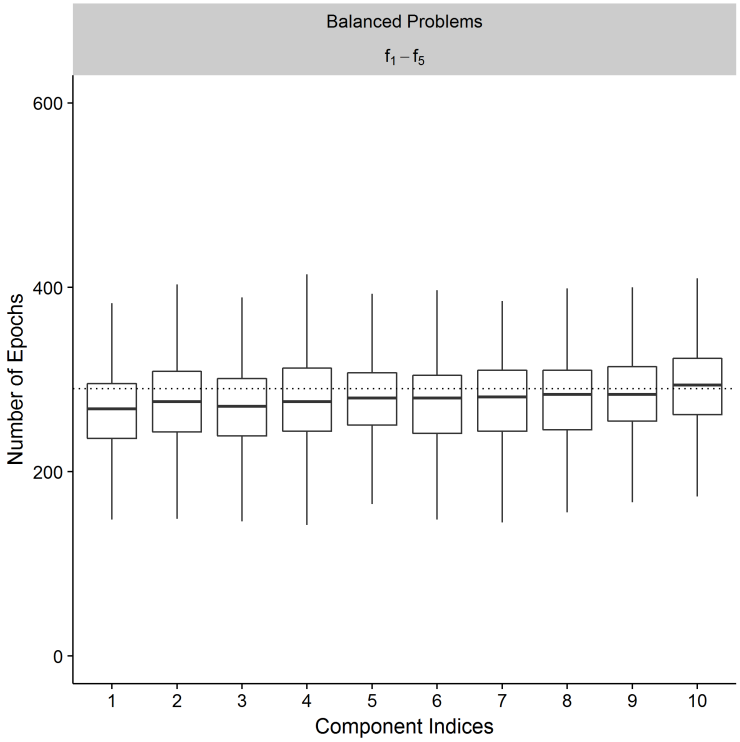


Figure 101: The performance of BBCC1 on balanced problems.

sizes, coefficients and base functions. Figure 101 presents the performance of BBCC1 on these problems.

The horizontal axis of Figure 101 represents the components indices and the vertical axis shows the number of epochs each component is selected. To produce a concise graphic, the statistics of similar components are aggregated. For example, the left-most box summarizes the average number of epochs that the first components of f_1–f_5 are selected. The dashed line in the figure indicates the performance of a round-robin CC (*i.e.,* 10% of available budget as $K = 1$).

As Figure 101 shows, all components of balanced problems are selected equally, regardless of their base functions. This confirms that in the case of uniform contributions, BBCC1 allocates the resources uniformly (the same as round-robin CCs). Therefore, applying BBCC framework on balanced problems should not degrade the performance of CC algorithms.

The Unequal Coefficients

In the second set of the case studies, we investigate the effect of unequal component coefficients using the second and third categories of the problems (*i.e.,* f_6–f_{15}). In contrast to the first group, the second and third categories consist of problems having components with nonuniform coefficients. The only difference between these two categories is the level of imbalance (*i.e.,* the variance in the coefficients' magnitude), in a sense that the imbalance in f_{11}–f_{15} are more severe than f_6–f_{10} (see Table 92 from Chapter 9).

Figure 102 presents the performance of BBCC1 on f_6–f_{15}. The figure consists of two facets, each of which belongs to one of the categories. In contrast to the balanced functions in Figure 101, we observe a marked increase in selection rates as the indices increase in Figure 102. Recall that the magnitude of a coefficient is an exponential function of the

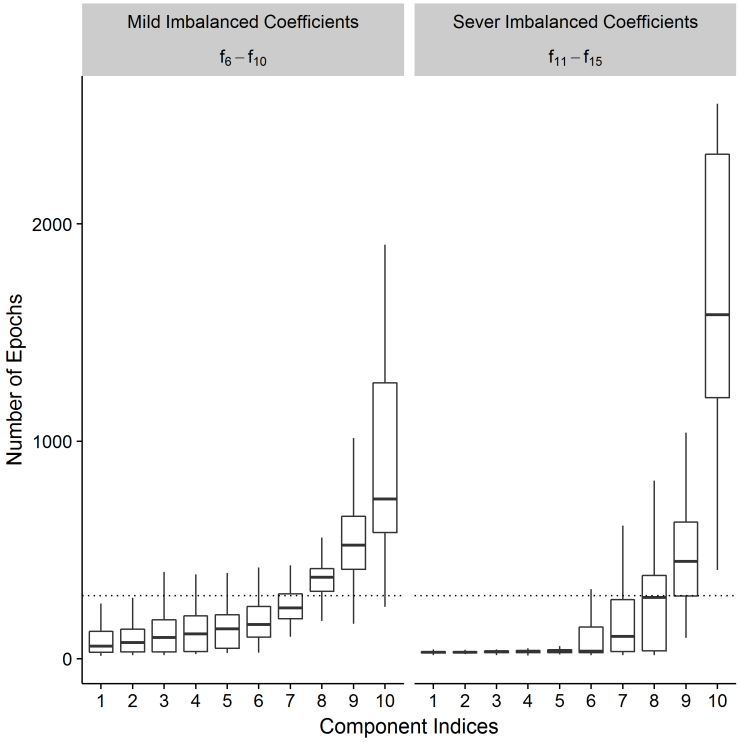


Figure 102: The effect of imbalance in component coefficients on the number of epochs a component is selected by BBCC for optimization.

component index (*i.e.,* 2^k for f_6–f_{10} and 10^k for f_{11}–f_{15}). Therefore, as we move towards the right side of a facet, the contributions increase.

The strong correlation between the coefficients and average number of selection approves that BBCC1 correctly identified the most contributing component among the others. A comparison between the BBCC1 resource allocation and traditional round-robin component selector (*i.e.,* the dashed line) reveals that the adaptive component selectors can advance the CCs outcome by allocating a larger portion of the resources to the most contributing components.

Another important observation from comparing categories with various levels of coefficient imbalance is the differences in the selection variances (height of a box plot is an indication of the variance in the corresponding population). Figure 101 shows that the variance in the selection among the first group of problems (f_1-f_5) is negligible and remains constant for all components. For imbalanced problems in Figure 102, however, the variance rapidly increases as the coefficients grow. Particularly for the severely imbalanced problems (f_{11}–f_{15}), the variances for the first five components (with the smallest coefficients) are almost zero. Nevertheless, this number for the last component (with the largest coefficient) is extremely large. To investigate whether the differences between search landscapes caused the large variances in the Figure 102, we compare two groups of problems with different base functions.

Figure 103 compares the behavior of BBCC1 on Rastrigin and Schwefel functions in three scenarios. As the plot shows, regardless of the base function, the selection ratio is constant for balanced functions (the solid lines for f_2 and f_4), and increasing for imbalanced functions (the dashed lines for f_7, f_9, f_{12} and f_{14}). However, the slope of the growth largely depends on the search landscape in the case of imbalanced functions. For

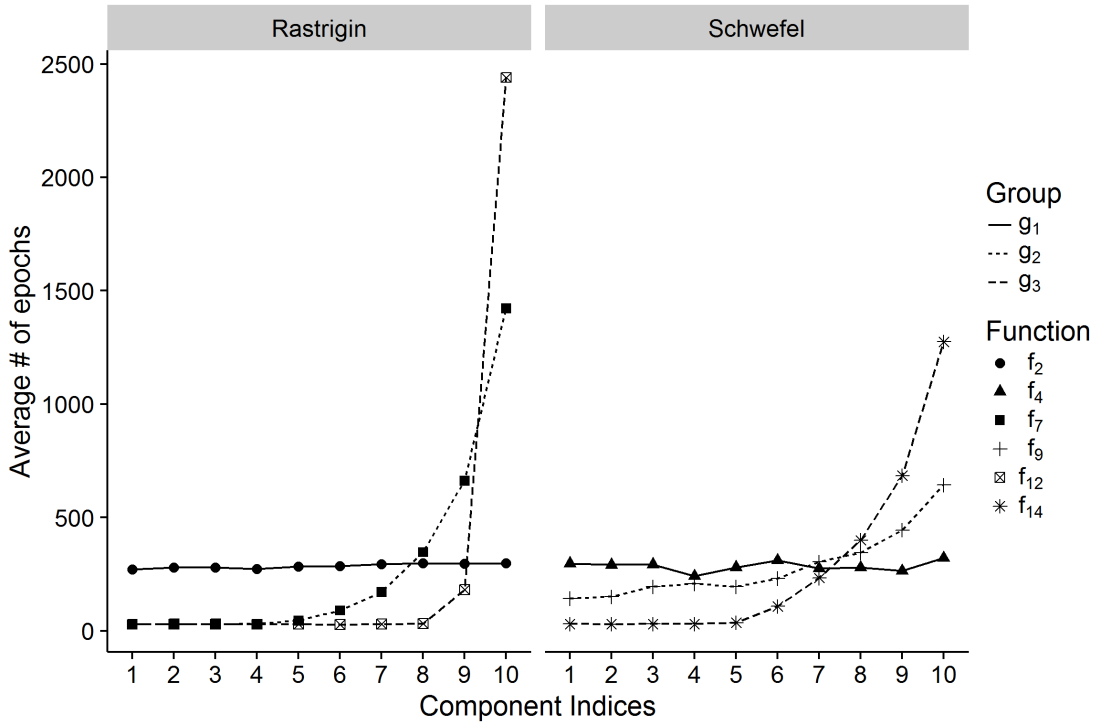


Figure 103: The effect of search landscape on the behavior of component selector when coefficients are imbalanced.

example, $f_{12,10}$ is selected almost 1000 times more often than $f_{14,10}$ which has the same coefficient and size, but different base function. This observation suggests that the base functions may affect the contribution of components. We will explore this topic further in Subsection 10.4.2.

The Unequal Dimensionality

In this part of the case studies, we investigate the impact of unequal component sizes using the fourth and fifth categories of problems (f_{16}–f_{25}). As presented in Table 92, all components of a problem in these categories have the same coefficients and base functions, while their sizes may vary. To create two problem sets with different imbalance levels, the ranges of subproblem sizes are deliberately chosen to be different in the fourth and fifth categories. Particularly, for f_{16}–f_{20} the component sizes are limited to 50, 150 and 200, while for f_{20}–f_{25} they vary from 25 to 250 (see Table 91 from Chapter 9).

Note that the nonuniform coefficients (studied in Subsection 10.4.2) only affects the importance (fitness range) of components, while varying component sizes also influence the complexity of the subproblems. This means, for f_{16}–f_{25} components with larger indices not only have a greater impact on the overall fitness values, but also tend to demand more objective function calls to be solved. Therefore, if a particular optimizer fails to make significant progress in the optimization of the large-size subproblems, the component selector may switch to components with more manageable sizes.

Figures 104 and 105 illustrate the average number of times that components with the same size are selected. As a general trend, components with higher dimensionality are selected more frequently regardless of the level of imbalance. This pattern is expected from CACCs as components with a larger number of decision variables tend to have a

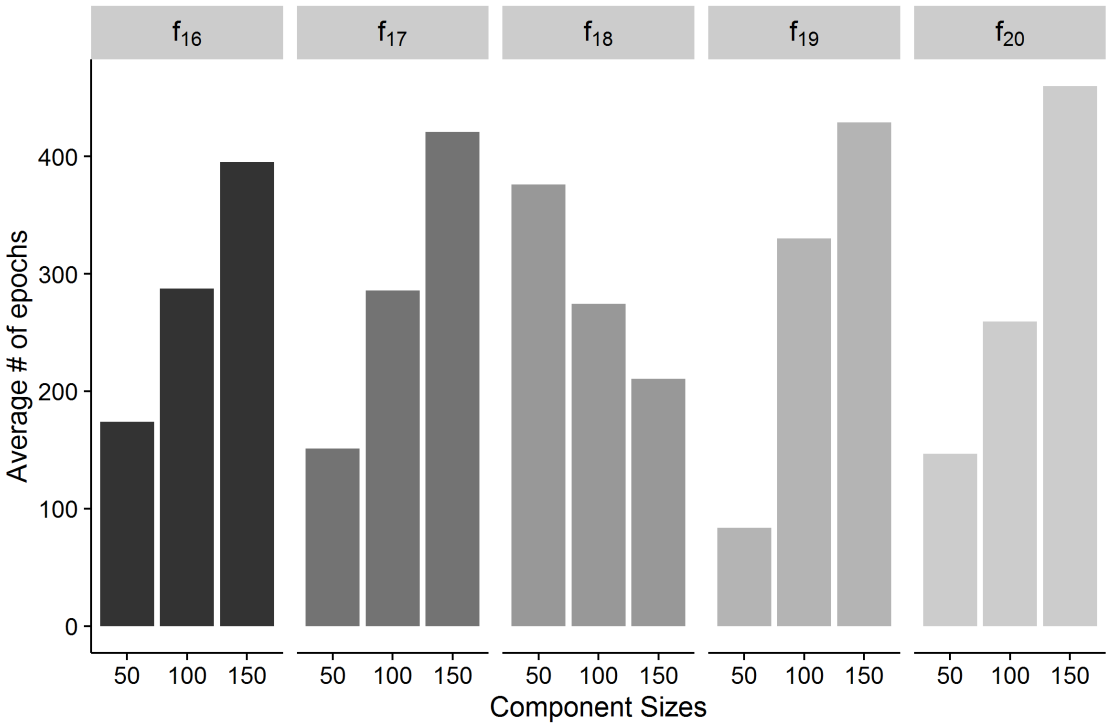


Figure 104: The effect of moderate imbalance in component sizes on the number of epochs a component is selected by BBCC for optimization.

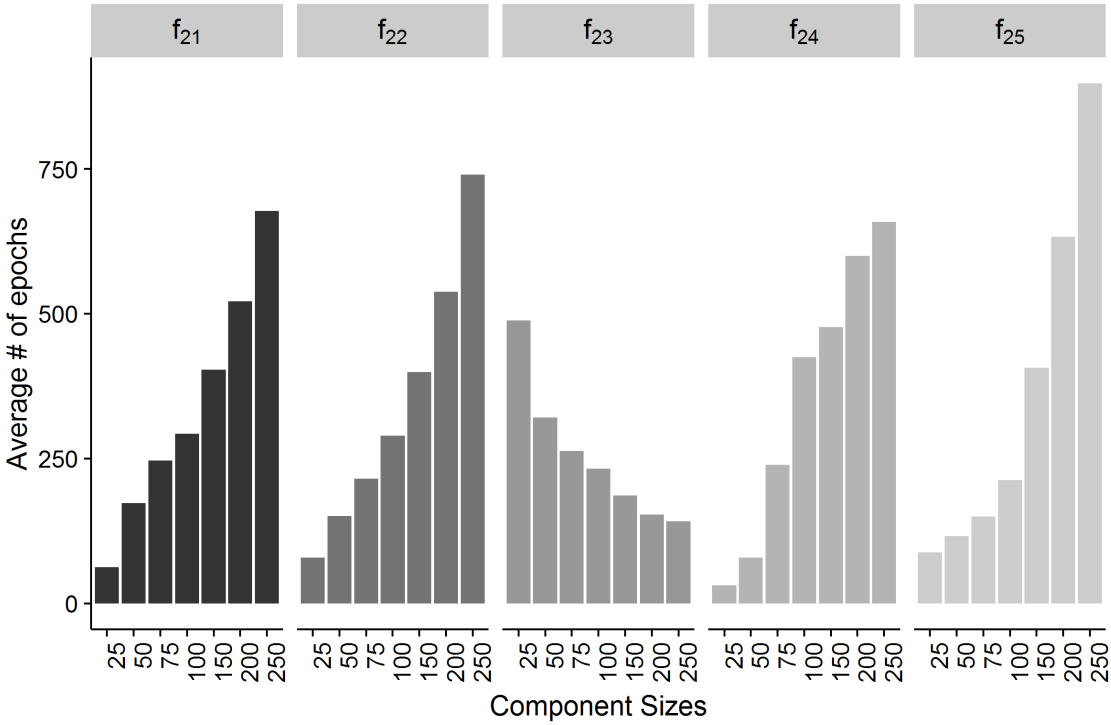


Figure 105: The effect of severe imbalance in component sizes on the number of epochs a component is selected by BBCC for optimization.

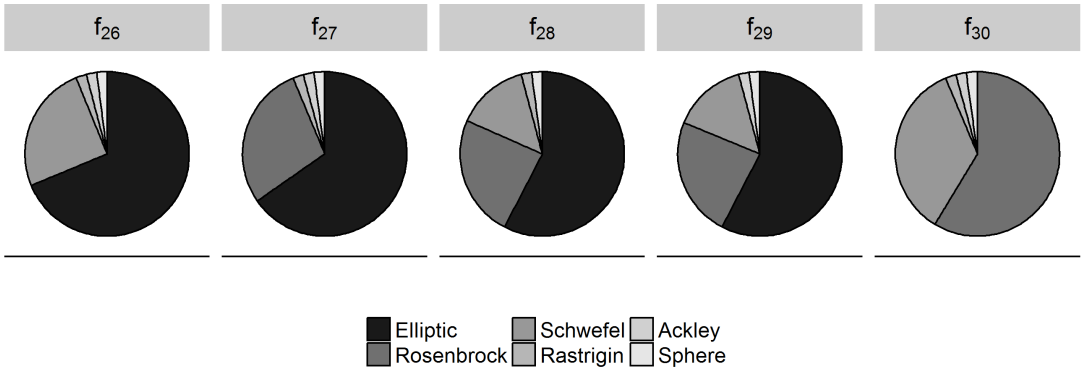


Figure 106: Average number of component selection.

stronger impact on the overall fitness than the other components.

The only exceptions in Figures 104 and 105 are Ackley's functions (f_{18} and f_{23}) which the smallest subproblems are selected the most. This behavior does not challenge the importance of dimensionality, but instead, it suggests that the basic DE/rand/1/bin is not powerful enough to improve very large Ackley's subproblems with the given budget. In these cases, the fitness improvements gained from optimizing smaller components are relatively larger than searching high-dimensional search spaces. As a result, BBCC1 selects the simpler subproblems more often than the higher dimensional components which our basic optimizer is incapable to improve significantly. This example highlights the effects of optimizer's power on the components relative contributions. Neglecting this factor may result in investing too much on optimization of components that are too complex for the adopted optimizer to obtain any significant improvement.

The Unequal Landscapes

To find out whether BBCC can differentiate between components with different landscapes, we apply it to the sixth category of problems, *i.e.,* f_{25}–f_{30}, each of which has a unique combination of five different base functions (see Table 94 and 93). To make the combinations different for each function, we leave out one of the six possible base functions. For example, f_{26} and f_{30} do not include Rosenbrock and Elliptic components, respectively. As mentioned in Subsection 9.3.2, all components of f_{26}–f_{30} have the same coefficient and size (*i.e.,* $\forall k, c_k = 1$ and $|\mathcal{D}_k| = 100$).

Each pie chart in Figure 106 corresponds to one of the problems and shows the distribution of selected components. Note that, each function consists of ten components with five different landscapes. Therefore, each pie has only five parts. We chose to use pie charts instead of line chart because, as opposed to the other problems, there is no meaningful relationship between the index and contribution of a component.

Figure 106 reveals that BBCC1 chooses components according to their base functions. For example, when Elliptic components exist in a problem (*i.e.,* f_{26}–f_{29}), they are selected more often than any other components such that at least 60% of the budget is allocated to them. After Elliptic, Rosenbrock, and Schwefel subfunctions are respectively the second and the third highly selected components, when they are included in a problem. On the other hand, Rastrigin, Ackley and Sphere base functions are the least selected components among the others such that only 2% of the budget is allocated to each of them.

Another interesting observation from Figure 106 is that both the simplest (*i.e.,* Sphere) and most difficult (*i.e.,* Ackley's) base functions, for this particular optimizer, are among

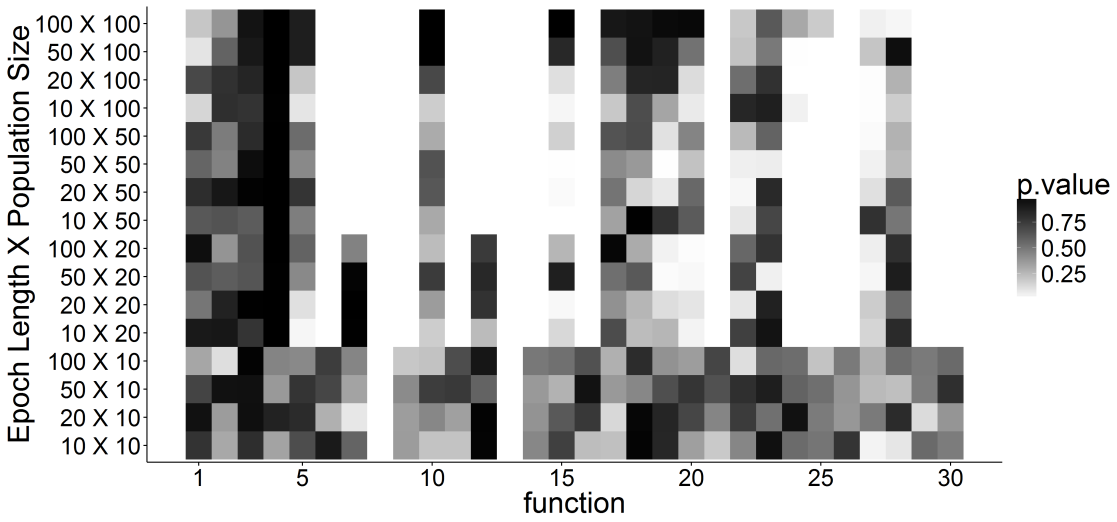


Figure 107: Comparing BBCC and CC performance in 16 different parameter settings. The brighter a tile is, the more likely that BBCC outperforms the CC with the same parameter values.

the least selected components. The rationale for this behavior is that since the Sphere subfunctions are easy to solve, they need very few epochs to reach a point that their contributions become relatively insignificant. In another extreme case, the 100-dimensional Ackley subfunctions are very difficult for our DE/rand/1/bin. In other words, given the same number of function calls, it is less likely to receive larger improvement over Ackley's subfunctions than the others. In such a case, BBCC1 selects components that bring the largest improvement for each single epoch. Note that adopting a different subfunction optimizer may change the order of selection and the portion of computational budget designated to each component. However, BBCC should still be able to adapt to the power of the adopted optimizer and effectively allocate the resources. Due to the limited space, we postpone further empirical studies on the adaptiveness of BBCC to an adopted optimizer to future work.

10.4.3 Sensitivity Analysis

In the previous part, we studied the performance of BBCC1 in dealing with different types of imbalance components, base functions, and imbalance levels. In this part, we analyze its sensitivity to the values of its parameters. Since BBCC can potentially have numerous instances and each having its own set of parameters, a comprehensive parameter analysis demands a separate study. Furthermore, a number of parameters are unique to some specific implementations. For example, ϵ and τ values are only meaningful in ϵ-greedy and Softmax component selectors, respectively. As a result, we choose the simplest implementation of BBCC (*i.e.,* BBCC1) to only study generic parameters that exist in all variants of this framework. Nonetheless, other implementations of BBCC may present different degrees of sensitivity to these generic parameters.

Here, we study the population size $N \in \{10, 20, 50, 100\}$ and epoch length $\delta_t \in \{10, 20, 50, 100\}$ which is the number of iterations each component is optimized when selected. Both of these parameters directly affect the budget allocated to a selected component. More precisely, a selected component consumes $N \times \delta_t$ function calls in each epoch. In this study, we compare BBCC1 and its round-robin counterpart, CC1, using the

same parameter setting on all the problems that we introduced in the case studies. The quality of final solutions obtained by these algorithms is compared using a one-sidedMann-Whitney U test with 95% confidence level where the H_1 is that the distribution of BBCC1 results is shifted to the left of the distribution of the CC outcomes. The resulting p-values are depicted in Figure 107. In this heatmap, a bright cell (*i.e.,* small p-value) indicates a strong superiority of the BBCC1 over CC1. The black cells, in contrast, determine both algorithms perform statistically similar.

The horizontal axis of the Figure 107 indicates the function indices and the vertical axis shows the $\delta_t \times N$. Therefore, the row at the bottom of the plot belongs to the epochs with the least allocated budget (only 100 function calls per epoch) and the row at the top associates with the epochs received the largest portion of the budget ($10,000$ function call per epoch). Since the total function calls are fixed, the number of epochs decreases as δ_t or N increases.

An observation from the figure is that the four rows at the bottom have the most number of dark cells. This suggests that $N = 10$ is not a good choice for this particular implementation of BBCC framework. Although having small population provides more exploration opportunities as the number of epochs increases, it seems DE/rand/1/bin cannot make any significant progress with this small population. Perhaps some other optimizers or improvement measures can remedy or alleviate this problem. In addition, having too many epochs forces a lot of component switches overhead to BBCC which can also contribute to its marginal improvement over CC in this particular case.

The cells in the five left-most columns are also dark colored which means BBCC1 and CC1 perform statistically comparable on f_1–f_5. This is not surprising as we already showed in the case studies that BBCC1 allocated budget uniformly when the problems are balanced (see Section 10.4.2). This shows that it is safe to use BBCC even if we are not sure whether a given problem is balanced or not.

For many of the problems with mild dimensional imbalance (especially f_{17}–f_{19}), BBCC1 demonstrates improvement over CC1 although they are not statistically significant. The most effective parameter settings for this category are $\delta_2 \in \{50, 100\}$ and $N = 20$.

The behavior of BBCC1 on Rosenbrock's and Ackley's problems is particularly interesting. In Rosenbrock's case, the more severe the imbalance the more likely BBCC1 outperforms CC1 (compare f_{15} with f_{10}, and f_{25} with f_{20}). It is also evident that BBCC1 handles the dimensional imbalance better than the coefficient imbalance in the Rosenbrock's case (compare f_{20} and f_{25} with f_{10} and f_{15}).

In Ackley's cases, BBCC1 easily outperforms CC1 on coefficient imbalance (*i.e.,* f_8 and f_{13}), however, it is less likely to improve the baseline in dimensionally imbalanced functions (*i.e.,* f_{18} and f_{23}). Once again, it confirms that DE/rand/1/bin is incapable of solving large-scale Ackley's functions when budget is limited.

For the last category of problems that includes heterogeneous subfunctions, BBCC1 improves CC1 performance in most of the cases as long as $N > 10$. Based on Figure 107, we can conclude that the most difficult case for BBCC1 is f_{28} followed by f_{27}. These problems consist of all base functions except Ackley's and Schwefel's functions, respectively. From Figure 106, we know that BBCC1 tends to spend most of the resources on Elliptic and Rastrigin's subfunctions when dealing with f_{27} and f_{28}

Figure 107 also suggests that the sensitivity to N may differ from one search landscape to another. For example, the effective range of N is considerably wider for Ackley's instances (*e.g.,* f_8 and f_{13}) than for Rastrigin's problems (*e.g.,* f_7 and f_{12}). The vertical blocks of four tiles (where N is fixed but δ_t varies) that are visible in several parts of Figure 107 is another indication of the sensitivity of BBCC1 (or DE) to population size.

There is almost no significant evidence in Figure 107 that suggests δ_t has a great influence on the relative performance of BBCC1. Indeed, in most of the cases, variation in δ_t has little or no effect on the p-values. In the other cases, it seems that having large δ_t when N is also large (*e.g.,* $N = 100$ and $\delta_t \geq 50$ for f_{10}, f_{15} and f_{17}–f_{20}) has an adverse effect on the relative performance of BBCC1. Here, the limited number of epochs can be the reason. Since the maximum number of objective function calls is fixed, having large values for these parameters results in a very few number of epochs.

Overall, this sensitivity analysis suggests that the behavior of BBCC1 may be affected by the landscape structure. Regardless of the base functions, the most effective range for N is $\{20, \cdots, 100\}$, while one should avoid choosing large values for both parameters at the same time. According to Figure 107, setting both values to 50 results in a significant improvement (BBCC1 over CC1) in the majority of instances.

10.4.4 BBCC vs. Round-robin CCs

In this part, we compare BBCC1 with two variants of round-robin CCs: CC1 and CC2. The main difference between these two CC algorithms is in the adopted optimizer; we use DE/rand/1/bin and SaNSDE as the subfunction optimizers in CC1 and CC2, respectively. For the comparisons, we use all partially separable imbalanced problems from CEC'13 LSGO benchmark set which are f_4-f_{11}. In addition, the effective decomposition for f_1–f_3 and f_{12}–f_{15} is unknown and existing decomposition techniques typically return a single component in which case using a CC framework becomes irrelevant.

As Table 102 shows, BBCC1 finds the best solutions in six out of eight cases. In the Ackley's instances (*i.e.,* f_6 and f_{10}), however, the improvements are not statistically significant. According to the Win-Tie-Loss (W-T-L) numbers, BBCC1 significantly outperforms each of the round-robin CCs in five problems. The Friedman ranks at the bottom of the table confirm that BBCC1 is the best performer among the compared CC variants.

10.4.5 BBCC vs. Contribution-aware CCs

In this part we conduct a series of comparison studies between BBCC1 and other CACCs such as CBCC [Omidvar et al. 2011; 2016], MOFBVE [Mahdavi et al. 2016c], and CCFR [Yang et al. 2017] variants.

Table 103 compares BBCC1 with CBCC1, CBCC2 and three variants of CBCC3. The Friedman ranks and orders show that BBCC1 achieves the best overall results in comparison with CBCCs, and closely followed by CBCC3 when $p_t \in \{0, 0.05\}$. The W-T-L numbers reveal that it significantly improves CBCC1 and CBCC2 in five out of eight problems while each of them outperforms BBCC1 in only one task. BBCC1 significantly improves CBCC3 variants in half of the cases, whilst performs statistically similar in two problems and losses in the other two.

The average nWins score of BBCC1 in Table 103 is +35%. These scores show that BBCC1 significantly outperforms all variants of CBCC in f_5, f_7, f_9 and f_{11}. It also improves all CBCC variants on Ackley's instances (*i.e.,* f_6 and f_{10}), although the improvements are not statistically significant. It is evident in Table 103 that the performance of BBCC1 on Elliptic (*i.e.,* f_4 and f_8) is lower than CBCC3 variants. Indeed, the fitness values highlighted in **bold** reveal that BBCC1 finds the best solutions for all problems except the Elliptic instances. Since CBCCs use an adaptive optimizer (*i.e.,* SaNSDE) rather than simple DE/rand/1/bin, one may conclude that adopting different optimizers is a contributing factor in the superiority of CBCCs in Elliptic cases. This hypothesis is supported by the fact that CC2 with SaNSDE also outperforms CC1 with DE/rand/1/bin

Table 102: BBCC1 vs. Round-robin CCs on CEC'13 LSGO Imbalanced Benchmarks. The μ and σ symbols represent mean and STD values, respectively. The best average fitness values are shown in bold.

function		BBCC1	CC1	CC2	nWins
f_4	μ	3.27e+08	1.52e+09	**1.97e+08**	**0**
	σ	1.41e+08	5.56e+08	1.51e+08	
f_5	μ	**1.45e+06**	4.20e+06	2.66e+06	**2**
	σ	3.23e+05	2.54e+06	7.12e+05	
f_6	μ	**1.04e+06**	1.04e+06	1.06e+06	**0**
	σ	1.47e+05	1.47e+05	1.49e+03	
f_7	μ	2.14e+05	**1.93e+05**	5.12e+07	**1**
	σ	1.56e+05	5.04e+04	3.67e+07	
f_8	μ	**6.96e+12**	3.02e+14	7.19e+13	**2**
	σ	4.21e+12	1.82e+14	6.07e+13	
f_9	μ	**1.21e+08**	5.69e+08	2.85e+08	**2**
	σ	3.42e+07	8.59e+07	6.20e+07	
f_{10}	μ	**9.21e+07**	9.28e+07	9.43e+07	**0**
	σ	1.30e+07	1.31e+07	3.64e+05	
f_{11}	μ	**5.99e+07**	5.68e+08	2.62e+10	**2**
	σ	6.64e+07	1.53e+08	3.10e+10	
W-T-L:		-	**5-3-0**	**5-2-1**	
rank:		**1.3125**	**2.3125**	**2.3750**	
order:		**1**	**2**	**3**	

in Elliptic problems (see Table 102). We intend to investigate this hypothesis by adopting a variety of optimizers on a wider range of problems in our future work.

Table 104 compares the results of BBCC1 with bi-level to 5-level MOFBVEs. Putting the special case of Ackley's functions (*i.e.,* f_6 and f_{10}) aside, BBCC1 significantly outperforms all variants of MOFBVE on all other imbalanced problems (except f_{11} for 5-level MOFBVE where there is a tie). Although the BBCC1's average nWins score is +46.87%, it shows no improvement over MOFBVE variants in the Ackley's cases. This is not a surprising observation as from Table 102 we recall that the BBCC1 was incapable of significantly improving the round-robin CCs in these problems.

As the W-T-L summary in Table 104 shows, BBCC1 performs better than MOFBVEs in the majority of the tasks. Indeed, it significantly improves each MOFBVE instance in at least six out of eight cases. In addition, the **bold** objective values in the table reveal that among all five algorithms, BBCC1 found the best solutions for five out of eight problems. Finally, the Friedman ranking identifies BBCC1 as the best algorithm.

Table 105 compares BBCC1 with three variants of CCFR. The main differences between these algorithms are the adopted grouping and optimizer. As the nWins scores (with the average of +50%) reveal, BBCC1 outperform all variants of CCFR on f_5, f_7, f_9, and f_{11}. However, its relative performance on Elliptic instances (*i.e.,* f_4 and f_8) is not promising. In these two cases, CCFR with CMA-ES optimizer performs the best.

The W-T-L numbers in Table 105 show that BBCC1 significantly improves CCFR with SaNSDE optimizer in six cases, and CCFR-CMA-ES in half of the cases. The

Table 103: BBCC1 vs. CBCCs on CEC'13 LSGO Imbalanced Benchmarks. The μ and σ symbols represent mean and STD values, respectively. The best average fitness values are shown in bold.

function		BBCC1	CBCC1	CBCC2	CBCC3			*nWins*
					$p_t = 0$	$p_t = 0.05$	$p_t = 1$	
f_4	μ	3.27e+08	7.71e+07	8.77e+10	**2.20e+07**	2.97e+07	4.08e+07	**-3**
	σ	1.41e+08	4.05e+07	1.14e+10	8.05e+06	1.56e+07	2.09e+07	
f_5	μ	**1.45e+06**	2.28e+06	2.09e+06	2.13e+06	1.99e+06	2.34e+06	**5**
	σ	3.23e+05	3.55e+05	3.52e+05	3.49e+05	3.61e+05	4.70e+05	
f_6	μ	**1.04e+06**	1.06e+06	1.06e+06	1.05e+06	1.05e+06	1.06e+06	**0**
	σ	1.47e+05	2.18e+03	1.65e+03	1.07e+04	1.97e+03	2.15e+03	
f_7	μ	**2.14e+05**	6.38e+07	8.82e+07	2.09e+07	1.42e+07	4.75e+07	**5**
	σ	1.56e+05	4.01e+07	6.78e+07	3.04e+07	2.18e+07	3.38e+07	
f_8	μ	6.96e+12	1.38e+13	1.88e+12	1.21e+10	**8.23e+09**	1.51e+11	**-3**
	σ	4.21e+12	1.14e+13	2.80e+11	2.40e+10	1.03e+10	2.87e+11	
f_9	μ	**1.21e+08**	2.32e+08	2.03e+08	1.40e+08	1.56e+08	2.02e+08	**5**
	σ	3.42e+07	4.85e+07	2.45e+07	1.55e+07	3.51e+07	5.09e+07	
f_{10}	μ	**9.21e+07**	9.41e+07	9.41e+07	9.22e+07	9.29e+07	9.40e+07	**0**
	σ	1.30e+07	3.91e+05	2.59e+05	1.10e+06	5.81e+05	4.82e+05	
f_{11}	μ	**5.99e+07**	1.58e+10	1.63e+10	4.74e+08	6.24e+08	1.33e+09	**5**
	σ	6.64e+07	2.26e+10	2.76e+10	2.95e+08	3.47e+08	1.41e+09	
W-T-L:		-	**5-2-1**	**5-2-1**	**4-2-2**	**4-2-2**	**4-2-2**	
rank:		**2.0625**	**5.1875**	**5.0625**	**2.2500**	**2.3125**	**4.1250**	
order:		**1**	**6**	**5**	**2**	**3**	**4**	

Friedman statistics rank BBCC1 as the best performer closely followed by CCFR-CMA-ES.

For further statistical analysis, we perform Friedman and Quade significant tests. Table 106 provides the obtained p-values. As the results show, both test show there is a significant differences between the performance of CBCCs and CCFRs when compared with BBCC1. Therefore, we perform pairwise posthoc test to find the pairs that are responsible for these differences. The results are summarized in Tables 107 and 108. Note that we do not perform posthoc tests on MOFBVE results since nor Friedman neither Quade test can detect a huge difference.

The Friedman posthoc test (*a.k.a.* Conover's pairwise tests) in Table 107 reveals that BBCC1 performs significantly better than CBCC1, CBCC2, and CBCC3 when $p_t = 1$. The pairwise posthoc-Quade test only confirms a significant difference between BBCC1 and the early variants of CBCC (*i.e.,* CBCC1 and CBCC2). In other cases, BBCC1's performance statistically comparable with CBCC3 variants.

According to Friedman's posthoc test in Table 108, BBCC1 performs significantly better than all CCFR variants except when CMA-ES is adopted as the optimizer. The posthoc-Quade tests, however, only confirms a significant difference between BBCC1 and CCFR-DG.

Overall, BBCC1, as the simplest implementation of BBCC framework, is ranked as the best performer when compared with the other available CACCs. As mentioned above,

Table 104: BBCC1 vs. MOFBVEs on CEC'13 LSGO Imbalanced Benchmarks. The μ and σ symbols represent mean and STD values, respectively. The best average fitness values are shown in bold.

function		**BBCC1**	**MOFBVE**				***nWins***
			bi-level	*3-level*	*4-level*	*5-level*	
f_4	μ	**3.27e+08**	9.09e+09	9.92e+09	9.13e+09	7.29e+09	**4**
	σ	1.41e+08	2.60e+09	5.01e+09	3.08e+09	3.63e+09	
f_5	μ	**1.45e+06**	2.69e+06	2.77e+06	2.80e+06	3.01e+06	**4**
	σ	3.23e+05	5.30e+05	4.19e+05	3.61e+05	5.43e+05	
f_6	μ	1.04e+06	**8.56e+04**	9.33e+04	9.25e+04	8.81e+04	**-4**
	σ	1.47e+05	2.41e+04	2.91e+04	2.15e+04	2.59e+04	
f_7	μ	**2.14e+05**	5.85e+06	5.82e+06	9.35e+06	6.53e+06	**4**
	σ	1.56e+05	2.18e+06	2.21e+06	1.26e+07	2.80e+06	
f_8	μ	**6.96e+12**	2.31e+13	1.81e+13	2.54e+13	2.88e+13	**4**
	σ	4.21e+12	8.86e+12	9.05e+12	1.03e+13	1.15e+13	
f_9	μ	**1.21e+08**	2.81e+08	2.65e+08	2.62e+08	1.94e+08	**4**
	σ	3.42e+07	3.09e+07	3.16e+07	2.62e+07	3.30e+07	
f_{10}	μ	9.21e+07	3.34e+04	2.09e+04	2.55e+03	**1.89e+03**	**-4**
	σ	1.30e+07	2.17e+04	2.22e+04	3.31e+02	1.15e+03	
f_{11}	μ	**5.99e+07**	7.64e+08	4.55e+08	4.48e+08	3.83e+09	**3**
	σ	6.64e+07	1.04e+09	3.66e+08	3.97e+08	1.34e+10	
W-T-L		-	**6-0-2**	**6-0-2**	**6-0-2**	**5-1-2**	
rank:		**2.000**	**3.125**	**3.250**	**3.375**	**3.250**	
order:		**1**	**2**	**3**	**4**	**3**	

the improvements are statistically sound, in many cases. We could probably observe more interesting patterns and detect more significant differences if the number of imbalanced benchmark functions was large enough.

10.4.6 BBCC vs. State-of-the-art Techniques

In the experiments, we compare the BBCC1 results with the two state-of-the-art algorithms: MA-SW-Chains [Molina et al. 2010] and MOS'13 [LaTorre et al. 2013]. These two particular algorithms are chosen because MA-SW-Chains is the winner of CEC'10 LSGO competition and MOS won the CEC'13 and CEC'15 LSGO competitions. The main goal of this brief comparison is to show that the simplest implementation of BBCC framework coupled with a very basic optimizer can compete with the most advanced large-scale optimizers. Since BBCC1 could outperform CC1 and other CACCs, we expect that by adopting these advanced EAs as the subfunction optimizer of BBCC instances, we can enhance their performance even further.

Table 109 presents the results of BBCC1, MA-SW-Chains [Molina et al. 2010] and MOS [LaTorre et al. 2013] on the imbalanced problems from CEC'13 LSGO benchmarks. As the results show, BBCC1 significantly outperforms MA-SW-Chains in five problems and MOS in only two cases. The MOS can outperform BBCC1 in four problems, whilst MA-SW-Chains cannot significantly beat BBCC1 in any cases.

Table 105: BBCC1 vs. CCFRs on CEC'13 LSGO Imbalanced Benchmarks. The μ and σ symbols represent mean and STD values, respectively. The best average fitness values are shown in bold.

function		**BBCC1**	**CCFR**			***nWins***
			IDG2	DG	CMA-ES	
f_4	μ	3.2e+08	9.6e+07	9.1e+10	**9.5e+07**	**-1**
	σ	1.4e+08	4.0e+07	5.6e+10	4.0e+07	
f_5	μ	**1.4e+06**	4.2e+07	3.0e+06	2.8e+06	**3**
	σ	3.2e+05	3.2e+05	5.2e+05	3.1e+05	
f_6	μ	**1.0e+06**	4.1e+07	1.1e+06	1.0e+06	**1**
	σ	1.4e+05	1.0e+03	1.6e+03	1.0e+03	
f_7	μ	**2.1e+05**	8.2e+08	1.4e+08	2.0e+07	**3**
	σ	1.5e+05	2.9e+07	9.7e+07	2.9e+07	
f_8	μ	6.eE+12	4.6e+11	1.6e+15	**6.6e+10**	**-1**
	σ	4.2e+12	9.5e+10	1.0e+15	9.5e+10	
f_9	μ	**1.2e+08**	8.1e+09	1.9e+08	1.8e+08	**3**
	σ	3.4e+07	2.8e+07	2.8e+07	2.8e+07	
f_{10}	μ	**9.2e+07**	7.9e+08	9.5e+07	9.4e+07	**1**
	σ	1.3e+07	1.8e+05	3.1e+05	1.8e+05	
f_{11}	μ	**5.9e+07**	1.4e+09	2.8e+10	4.1e+08	**3**
	σ	6.6e+07	3.4e+08	6.0e+10	3.4e+08	
W-T-L:		-	**6-0-2**	**6-2-0**	**4-2-2**	
rank:		**1.500**	**3.375**	**3.375**	**1.750**	
order:		**1**	**3**	**3**	**2**	

Table 106: Overall Significance Tests for BBCC1 vs. Contribution-awar CCs. All p-values smaller than 0.05 are shown in bold.

test	**CBCCs**	**MOFBVEs**	**CCFRs**
Friedman	**1.3e-04**	3.92e-01	**1.94e-03**
Quade	**1.8e-05**	1.48e-01	**7.13e-03**

The average nWins score of BBCC1 is positive (+18.75%) and it can achieve the second position based on Friedman ranking. However, none of the statistical tests confirm any significant differences between BBCC1 and the winners of the previous LSGO competitions. The Friedman and Quade p-values for the results in Table 109 are 3.24e−01 and 4.08e−01, respectively. These statistics confirm that the most basic implementation of BBCC can compete with the most advanced optimizers on solving large-scale imbalanced optimization problems.

10.5 Chapter Summary

In this chapter, we have proposed a general framework called bandit-based cooperative coevolution (BBCC) to address the imbalance subproblem contributions in large-scale op-

Table 107: Pairwise Significance Tests on BBCC1 and CBCCs. All p-values smaller than 0.05 are shown in bold.

test	*adjustment*	**CBCC1**	**CBCC2**	**CBCC3**		
				$p_t = 0$	$p_t = 0.05$	$p_t = 1$
Friedman	none	**1.6e-06**	**3.2e-06**	7.3e-01	6.4e-01	**5.5e-04**
	Holm	**8.0e-06**	**1.2e-05**	7.3e-01	1	**1.6e-03**
	Bonferroni	**8.0e-06**	**1.6e-05**	1	1	**2.7e-03**
Quade	none	**4.0e-03**	**4.4e-03**	3.8e-01	5.2e-01	2.2e-01
	Holm	**2.0e-02**	**1.7e-02**	7.7e-01	5.2e-01	6.6e-01
	Bonferroni	**2.0e-02**	**2.2e-02**	1	1	1

Table 108: Pairwise Significance Tests on BBCC1 and CCFR. All p-values smaller than 0.05 are shown in bold.

test	*adjustment*	**CCFR**		
		IDG2	DG	CMA-ES
Friedman	none	**1.5e-05**	**1.5e-05**	4.6e-01
	Holm	**3.7e-05**	**3.75-05**	4.6e-01
	Bonferroni	**4.5e-05**	**4.5e-05**	1
Quade	none	5.9e-02	**6.9e-03**	6.7e-01
	Holm	1.1e-01	**2.0e-02**	6.7e-01
	Bonferroni	1.7e-01	**2.0e-02**	1

Table 109: BBCC vs. State-of-The-Art LSGO Algorithms [LaTorre et al. 2015] on CEC'13 LSGO Benchmark. μ and σ represent mean and STD values, respectively.

		BBCC	**CCCMAES**	**MASWC**	**2SEnsemble**	**MOS11**	**MOS12**	**MOS13**	**nWins**
f_4	μ	2.93E+08	2.82E+09	3.80E+09	3.80E+09	1.34E+10	3.07E+08	**8.73E+07**	**4**
	σ	1.00E+08	1.84E+09	2.70E+09	2.70E+09	7.69E+09	1.47E+08	3.11E+07	
f_5	μ	**1.05E+06**	7.28E+14	2.26E+06	2.26E+06	1.11E+07	2.41E+07	6.89E+06	**6**
	σ	1.84E+05	5.18E+06	1.36E+06	1.36E+06	1.79E+06	3.93E+06	9.16E+05	
f_6	μ	1.01E+06	1.74E+05	1.07E+04	1.94E+05	9.85E+05	9.89E+05	**1.43E+05**	**-6**
	σ	2.69E+04	2.09E+04	2.09E+04	1.64E+04	3.22E+03	2.67E+03	6.86E+04	
f_7	μ	**7.35E+01**	1.02E+09	3.78E+06	1.90E+06	2.31E+07	1.46E+05	4.65E+03	**6**
	σ	5.21E+02	4.89E+08	8.46E+05	1.14E+06	4.42E+07	1.12E+05	1.06E+04	
f_8	μ	1.36E+13	6.94E+15	4.63E+13	3.85E+14	1.64E+15	**7.18E+11**	2.85E+12	**2**
	σ	6.45E+12	3.37E+15	9.18E+12	1.39E+14	1.66E+15	5.42E+11	1.44E+12	
f_9	μ	**7.20E+07**	5.47E+08	1.14E+08	1.31E+08	8.97E+08	1.65E+09	3.99E+08	**6**
	σ	1.75E+07	8.94E+07	2.05E+07	1.51E+07	1.39E+08	3.47E+08	6.26E+07	
f_{10}	μ	9.42E+07	2.43E+07	3.66E+04	1.43E+07	6.65E+07	9.00E+07	**9.38E+05**	**-6**
	σ	2.34E+06	9.75E+06	6.17E+04	2.87E+06	2.91E+07	5.04E+05	4.79E+05	
f_{11}	μ	**3.31E+06**	1.24E+08	2.10E+08	2.38E+08	4.01E+10	2.71E+07	1.73E+07	**6**
	σ	1.15E+07	9.88E+07	2.43E+07	6.45E+07	1.23E+11	5.19E+06	5.04E+06	
Win-Tie-Loss		**7-0-1**	**6-0-2**	**6-0-2**	**5-1-2**	**4-1-3**	**4-0-4**		

timization problems while avoiding the pitfalls of the previously proposed CACCs. In contrast with the traditional CC framework which uniformly allocates computational resources to all components, BBCC adopts well-studied bandit algorithms to learn the contribution of each component to the long-term improvement in objective value and allocate resources accordingly.

Through extensive experiments on 30 imbalance large-scale benchmarks, we have shown that BBCC has the ability to handle a variety of scenarios. The sensitivity studies revealed that our simple BBCC implementation is robust with respect to the changes in generic parameter values. Our comparison studies also confirmed that the efficiency of BBCC in exploration-exploitation maintenance helps it to outperform previous budget allocation techniques used in CCs. We have also demonstrated that when some degrees of imbalance exists in the problems, even simple instances of BBCC perform statistically similar or better than the state-of-the-art algorithms that ignore such property of the problems.

With respect to the flexibility and generality of the proposed framework, we expect several future studies to analyze BBCC's performance in different scenarios, to expand the framework, and to apply it to real-world applications. There are some interesting studies that we left for the future as each one of them requires massive amount of experiments, analysis, and discussions which is obviously out of the scope of this research. In future, we will further examine the sensitivity of BBCC to the accuracy of decomposition and the number of components, study the behavior of BBCC when a dynamic grouping algorithm is adopted, and compare different instances of BBCC in order to find the most effective combination of the improvement measure, contribution estimator, and component selector algorithms.

CHAPTER 11

Conclusion and Future Work

In this dissertation, we studied two major approaches to enhance the use of limited computational resources in solving large-scale black-box optimization problems. The first approach was to make the best use of the very first step of every population-based metaheuristics: population initialization.

We started by reviewing and categorizing virtually all possible initialization algorithms and study the advantages and disadvantages of each family of them. Then, we investigated several directions to increase the functionality of the solver in hand just by using an alternative point generation method. Our experiments confirmed that in some cases such a small change could significantly improve the quality of the final solution. Moreover, we identified a few influencing factors such as the optimality of optimizers key parameters, population size, and the ability of initializer to generate uniformly scattered points in high-dimensional spaces. Our experiments suggest that the magnitude of the impact of these factors may change as dimensionality increases.

The second approach towards more effective use of limited computational resources that we studied here was targetting imbalanced problems. Often, the real-world optimization tasks can be decomposed into smaller subproblems which are easier to solve. Traditionally, all of these problem components are treated equally regardless of their dimensionality and complexity. Through several experiments, we showed that uniformly distributing the scarce computational budget across unequal subfunctions is a waste of resources. Therefore, we proposed several algorithms to identify the contribution of optimizing each black-box subproblem in the quality of the final solution and adaptively adjust the budgeting schema based on the estimated contributions. Our extensive numerical simulation on an extended benchmark especially designed for studying the resource allocation to imbalanced subproblems showed significant improvement over baselines and state-of-the-art techniques.

In the following, we revisit the key research objectives that we established in Chapter 1 and summarize what we have done to achieve each of them. Then, we discuss potential directions for future work, and finally, present the concluding remark of the thesis.

11.1 Research Objectives Revisited

1. To collect, study, survey, and categorize the published scientific articles on population initialization techniques that have been used in metaheuristics.

In Chapter 3, we reviewed the major population initialization approaches and then categorized them from three aspects: randomness (*i.e.,* based on their tendency to produce static or stochastic pointsets), compositionality (*i.e.,* how they can be broken into their building-blocks), and generality (*i.e.,* being designed for a specific task or being a general-purpose tool). This multifacet taxonomy expanded our understanding of the similarities and differences between the available initializers. Besides, it helped us in setting the practical experiments in the following chapters more uniformly and inclusively such that we have enough representative methods from each class of initialization algorithms.

2. To assess the quality of initial populations generated by different techniquesas the number of decision variables grows.

We investigated the effectiveness of several population initialization techniques with different population sizes on small, medium, and large-scale problems in Chapter 4. To maximize the generalization capacity of the experiments, we included multiple representatives from different initializer categories that we previously proposed in Chapter 3.

We empirically showed that when the computational resources are scarce, we can improve the final solution quality by merely using a different set of initial points. The conclusion is still valid regardless of the problem dimensionality, although the amount of gained improvement may decrease as the number of decision variables grows. We also found that different instances of one class of initializers may perform statistically different. Therefore, no category could be identified as the most effective class of initializers among the studied categories.

In Chapter 5, we examined the effect of parameter settings on the performance of several population initializers. The obtained results showed that when the values of the key parameters are not tunes (*i.e.,* commonly used values are adopted), the difference between the performance of initialization algorithms is statistically significant. However, when the optimum values for each parameter are employed[1], the advantages of adopting the alternative initialization techniques fades.

To investigate the main reasons of performance drop for some of the initialization techniques in high-dimensional spaces, we conducted an extensive set of experiments in Chapter 6. The hypothesis was that these techniques might struggle to produce evenly scattered set of points in higher dimensions. To assess the hypothesis, we used generic uniformity measures to study the effects of dimensionality (from $D = 2$ to $D = 1{,}000$) and population size (from 10 to 10,000 points) on conventional and alternative initialization techniques.

Our investigations confirmed that the uniformity of the initial population exponentially drops when dimensionality rises linearly. We showed that the low uniformity, weak coverage, and low diversity degrade the quality of populations dramatically even by significantly increasing the population size.

3. To study the mutual effects of population initialization and optimizer parameters on the performance of metaheuristics when the computational resources are limited.

We conducted a series of experiments in Chapter 5 to thoroughly investigate the effectiveness of high-dimensional initial population in two scenarios: 1) when default values are used for the optimizers parameters, and 2) when the parameters are fine-tuned to maximize the quality of the final solution. Our analysis demonstrated that just by using a different population initializer, we could achieve significantly better performance in the first scenario. In contrast, contemporary initializers can only marginally enhance the effectiveness of the metaheuristics in dealing with large-scale problems when the parameters

[1]Note that extensive experiments need to be run to identify these values for each optimization problem.

are finely tuned. Our findings suggest that when the number of objective function evaluations is fixed, tunning population size is as important as adopting powerful initialization technique.

4. To analyze and improve the robustness of the contribution-aware cooperative coevolutionary techniques in dealing with imbalanced problems.

In Chapter 7 we discussed that how the *curse of imbalanced contribution* affects the functionality of cooperative coevolutionary algorithms in solving large-scale black-box problems. We surveyed the previously proposed contribution-aware techniques that allocate the limited resources more efficiently by assigning a portion of the budget to subproblem based on its estimated impact in solving the main optimization task. We also assessed the complexity and accuracy of the available techniques in the calculation of components contribution as well as the effectiveness of their budget allocation schema.

We conducted a series of experiments in Chapter 8 to analyze the influence of decomposition accuracy and the severity of contribution imbalance on the performance of two well-known contribution-aware techniques. Our investigations revealed that, in general, contribution-based algorithms outperform traditional uniform resource allocation schema. However, we found that these techniques cannot maintain a robust balance between exploration and exploitation in the component space. In other words, they either spend too many resources on the subproblems they found contributing from the very early generations (which may have a reduced contribution over time) or wasting too many optimization iterations on exploring other components to see if their contribution changed.

In Chapter 8 we proposed a new algorithm to improve the budget allocation strategy of the studied techniques. Our case studies confirmed that by eliminating the far past signals of contribution from the equation and controlling the balance between component exploration and exploitation, we could significantly enhance the performance of the contribution-aware algorithm.

5. To design and implement a comprehensive set of large-scale imbalanced problems and benchmark the contribution-aware techniques on the proposed problem suite.

In Chapter 7 we argued that the previous studies analyzed the effectiveness of the contribution-aware in very limited scenarios. To avoid such pitfall, we expanded the available benchmarking set in Chapter 8 to cover other more realistic situations such as imperfect decomposition and functions with different levels of contribution imbalance. This new suite provided a more in-depth insight about contribution-aware techniques, but still could be expanded. Therefore, in Chapter 9, we systematically designed the most comprehensive set of 40 separable large-scale problems that covers eight categories of contribution imbalance, each of which reflects a unique scenario.

Besides, we benchmarked some of the round-robin and contribution-aware cooperative coevolutionary optimizers using the proposed set of testing functions. The results of our comparison studies confirmed that the new testbed provided several valuable insights about the studied techniques that were not possible to gain using the limited number of cases in the previously proposed test sets.

6. To formulate economical resource allocation problem as a dynamic multi-armed bandit task, and propose a general cooperative coevolutionary framework to tackle large-scale imbalanced problems more efficiently.

To address the shortcomings of the traditional uniform budget assignment in cooperative co-evolutionary as well as improving the functionality of contribution-aware techniques, we designed a new generic framework based on multi-armed bandit techniques in Chapter 10.

Through extensive experimental studies, we showed that our framework could handle a variety of cases while maintaining its robustness with respect to its parameter configuration. We showed that even when the contribution imbalance is minor, even the most basic implementation of our bandit-based framework can provide statistically similar or better results than the most advanced state-of-the-art large-scale optimizers. Our in-depth investigations revealed that maintaining the exploration-exploitation balance in component space is the critical factor of our framework's superior performance compared to the other contribution-aware techniques.

11.2 Future Work

In this thesis, we took an important step towards more efficient use of restricted computational budget in solving black-box large-scale continuous optimization problems. Nevertheless, there remain some other aspects that worth conducting further researches. In what follows, we list and discuss some of these potentials.

11.2.1 Scalable Population Initialization Techniques

We devoted the first part of this dissertation to assess and elaborate on the importance of scalable population initialization techniques in solving black-box problems when the resources are limited. Besides what we have studied, we identified the following areas for future investigations:

- As pointed out in Chapter 3, the majority of research in this domain is limited to only single-objective, box-bounded, and continuous problems. Nevertheless, there are many real-world applications such as scheduling, shortest path, and machine allocation that are multi-objective, tightly constrained, and discrete problems. Considering the complexity of these large-scale tasks and the scarcity of computational budget, leveraging on a more useful set of initial solutions can significantly improve the performance of solvers. Although to some extent our findings can be applied to these tasks, still a lot more can be done in this area.

- The magnitude of the best objective value after spending a fixed number of 3,000,000 objective function calls is the only metric used in the majority of experimental studies on large-scale problems (see Chapters 4 and 5). In many cases, especially when comparing various population initialization techniques, the contrast between the algorithms is evident at the beginning but fades out as the optimization progresses. In some other cases, the optimization procedures are stopped due to limited computational budget before the population converges. What is missing here is the study of optimizers in different budgeting scenarios. For example, some uniform population initializers may perform very well in cases that the resources are very limited (*e.g.,* 3,000 function calls) whereas some other techniques may continue improving the objective values if the budgeting is not very tight (*e.g.,* 3,000,000,000 function calls). Therefore, we need to study the optimizers in more than one fixed scenario (see Chapters 6, for example). Moreover, other metrics such as robustness of the outcome in different trials need to be measured, reported, and investigated.

- Although researchers on population initializers and cooperative coevolution increased in recent years, we could not find many significant studies on the combination of these two popular approaches. An interesting idea worth exploring is to postpone the population initialization to post problem decomposition stage and then using

different initializers for each subpopulation. These unequal initial subpopulations can address the component contribution imbalance issue that we explored in the second part of the thesis. Another thought would be to measure the effect of initial population on the accuracy and efficiency of problem decomposer. Note that, the decision variable grouping techniques consume a considerable portion of resources before the optimization process starts. Therefore, we might be interested in investigating whether more uniformly scattered initial points can lead to a less expensive problem decomposition.

11.2.2 Effective Budget Allocation in Cooperative Coevolution

We dedicated the second part of this thesis on designing more effective budget allocation techniques for cooperative coevolution algorithms. We showed that they are very successful primarily when the subproblems represent unique features and hence demand unequal portions of resources. Some related research ideas left unexplored are listed below.

- In Chapter 9, we proposed a comprehensive testbed covering eight unique imbalanced contribution scenarios in the context of large-scale modular problems. We comprehensively analyzed four optimization techniques based on this massive testbed. Conducting similar studies to benchmark other large-scale optimization algorithms can provide invaluable insight into how they perform in such realistic scenarios that are missing in the previous benchmarking sets.

- The generic bandit-based budget allocation framework that we proposed in Chapter 10 is comprised of four building blocks: component pool, improvement measure, contribution estimator, and component selector. Except for the component pool that is imposed by the problem decomposer or domain expert(s), the other can have multiple instances with a wide range of attributes. Different combinations of these building-blocks will result in numerous unique instances of our flexible framework. In this thesis, we discussed some of the possibilities and explored one variant in detail. Implementing, analyzing, and comparing other instances will open doors to new research opportunities in this research area.

- With some small modification, one can expand the bandit-based cooperative coevolution framework (discussed in Chapter 10) to a new dynamic decomposition technique. The new technique can address the long-standing problem of finding the most effective decomposition for fully separable functions. As mentioned in Chapter 2, splitting a function into many small subproblems may not lead to the best result, especially when the computational resources are limited. Therefore, an online resource allocation technique similar to our proposed framework can explore different combinations of small subfunctions and create clusters with optimum sizes to minimize the overhead of context switching and context vector reconstruction expenses.

- As we empirically showed in Chapters 8 and 10, compared to traditional cooperative coevolutionary algorithms, the contribution-aware techniques spend the resources more efficiently. However, they still invest a large amount of budget on exploring the component space. We believe the predictive power of surrogate-assisted models can help to reduce the waste of resources by a significant magnitude. As an example, we can train a regressor based on the historical observations of components performance to predict their future contribution if they are selected. Then, we can use these

estimates in ranking and selecting the next subproblem to be optimized. Note that the component space is decidedly smaller than search space (*e.g.,* we may group 1,000 variables into only ten components). Therefore, the predictive models, in this case, can achieve higher accuracy while using less training samples compared with the traditional surrogate-assisted models that try to learn the very vast and complex search landscape.

11.3 Concluding Remark

The curse of dimensionality and the practical limitations on the computational budget reduce the efficiency of the metaheuristics in solving large-scale problems dramatically. Therefore, making the best use of limited resources becomes critical in such cases. In this thesis, we have explored two main approaches towards more efficient use of computational resources: 1) improving the exploration power of population-based optimizers by adopting better population initialization techniques, and 2) more effective resource allocation in cooperative coevolutionary algorithms when a high-dimensional problem is decomposed into heterogeneous subproblems. The obtained experimental results confirmed that both strategies could significantly improve the performance of optimizers in a variety of scenarios.

Bibliography

H. A. Abbass. The self-adaptive pareto differential evolution algorithm. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 1, pages 831–836. IEEE, 2002.

L. Ai, M. Tang, and C. Fidge. Resource allocation and scheduling of multiple composite web services in cloud computing using cooperative coevolution genetic algorithm. In *Neural Information Processing*, pages 258–267. Springer, 2011.

B. Akay and D. Karaboga. Artificial bee colony algorithm for large-scale problems and engineering design optimization. *Journal of intelligent manufacturing*, 23(4), 2012.

F. S. Al-Qunaieer, H. R. Tizhoosh, and S. Rahnamayan. Opposition based computing—a survey. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pages 1–7. IEEE, 2010.

M. Ali, M. Pant, and A. Abraham. Unconventional initialization methods for differential evolution. *Applied Mathematics and Computation*, 2012.

Y. M. B. Ali. Soft adaptive particle swarm algorithm for large scale optimization. In *Bio-Inspired Computing: Theories and Applications (BIC-TA), 2010 IEEE Fifth International Conference on*, pages 1658–1662. IEEE, 2010.

J. Arellano-Verdejo, R. Barron-Fernandez, and H. Taud. Micro differential evolution performance empirical study for high dimensional optimization problems. In *Large-Scale Scientific Computing: 9th International Conference, LSSC 2013, Sozopol, Bulgaria, June 3-7, 2013. Revised Selected Papers*, volume 8353, page 281. Springer, 2014.

Y. Atay, I. Koc, I. Babaoglu, and H. Kodaz. Community detection from biological and social networks: A comparative analysis of metaheuristic algorithms. *Applied Soft Computing*, 50:194–211, 2017.

A. Auger and N. Hansen. A restart CMA evolution strategy with increasing population size. In *IEEE Congress on Evolutionary Computation (CEC'05)*, volume 2, pages 1769–1776. IEEE, 2005. G-CMA-ES.

T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms.* ser. Dover Books on Mathematics. Oxford University Press, 1996.

S. Bahmann and J. Kortus. EVO–evolutionary algorithm for crystal structure prediction. *Computer Physics Communications*, 184(6):1618–1625, 2013.

J. E. Beasley. *Advances in linear and integer programming.* Oxford University Press, Inc., 1996.

R. Bellman. Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences*, 42(10):767–769, 1956.

J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische mathematik*, 4(1):238–252, 1962.

D. P. Bertsekas, A. Nedi, A. E. Ozdaglar, et al. *Convex analysis and optimization*. Athena Scientific, 2003.

M. Bhattacharya, R. Islam, and J. Abawajy. Evolutionary optimization: a big data perspective. *Journal of network and computer applications*, 59:416–426, 2016.

A. S. Bondarenko, D. M. Bortz, and J. J. Moré. Cops: Large-scale nonlinearly constrained optimization problems. *Mathematics and Computer Science Division, Argonne National Laboratory, Technical Report ANL/MCS-TM-237*, 1999.

P. A. Bosman and D. Thierens. Expanding from discrete to continuous estimation of distribution algorithms: The id\mathbb {E} a. In *International Conference on Parallel Problem Solving from Nature*, pages 767–776. Springer, 2000. IDEA.

A. Bouaricha and J. J. Morè. Impact of partial separability on large-scale optimization. In *Computational Issues in High Performance Software for Nonlinear Optimization*, pages 27–40. Springer, 1997.

I. Boussaïd, J. Lepagnot, and P. Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237:82–117, 2013.

P. Bratley and B. L. Fox. Algorithm 659: Implementing sobol's quasirandom sequence generator. *ACM Transactions on Mathematical Software (TOMS)*, 14(1):88–100, 1988.

J. Brest and M. S. Maučec. Population size reduction for the differential evolution algorithm. *Applied Intelligence*, 29(3):228–247, 2008.

J. Brest and M. S. Maučec. Self-adaptive differential evolution algorithm using population size reduction and three strategies. *Soft Computing*, 15(11):2157–2174, 2011.

J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *Evolutionary Computation, IEEE Transactions on*, 10(6):646–657, 2006.

J. Brest, A. Zamuda, B. Boskovic, M. S. Maucec, and V. Zumer. High-dimensional real-parameter optimization using self-adaptive differential evolution algorithm with population size reduction. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 2032–2039. IEEE, 2008.

K. K. Budhraja, A. Singh, G. Dubey, and A. Khosla. Exploration enhanced particle swarm optimization using guided re-initialization. In *Proceedings of seventh international conference on bio-inspired computing: theories and applications (BIC-TA 2012)*, pages 403–416. Springer, 2013.

E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

J. P. Burkett. *Microeconomics: optimization, experiments, and behavior*. Oxford University Press, 2006.

Y. Carson and A. Maria. Simulation optimization: methods and applications. In *Proceedings of the 29th conference on Winter simulation*, pages 118–126. IEEE Computer Society, 1997.

U. K. Chakraborty. *Advances in differential evolution*, volume 143. Springer, 2008.

A. Chatterjee, S. Ghoshal, and V. Mukherjee. Solution of combined economic and emission dispatch problems of power systems by an opposition-based harmony search algorithm. *International Journal of Electrical Power & Energy Systems*, 39(1):9–20, 2012.

L.-L. Chen, C. Liao, W. Lin, L. Chang, and X.-M. Zhong. Hybrid-surrogate-model-based efficient global optimization for high-dimensional antenna design. *Progress In Electromagnetics Research*, 124:85–100, 2012.

W. Chen and K. Tang. Impact of problem decomposition on cooperative coevolution. In *IEEE Congress on Evolutionary Computation (CEC'13)*, pages 733–740. IEEE, 2013.

W. Chen, Y. Yuan, and L. Zhang. Scalable influence maximization in social networks under the linear threshold model. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 88–97. IEEE, 2010.

W. Chen, T. Weise, Z. Yang, and K. Tang. Large-scale global optimization using cooperative coevolution with variable interaction learning. In *Proc. of International Conference on Parallel Problem Solving from Nature*, volume 6239 of *Lecture Notes in Computer Science*, pages 300–309. Springer Berlin / Heidelberg, 2011a. CCVIL.

X. Chen, Y.-S. Ong, M.-H. Lim, and K. C. Tan. A multi-facet survey on memetic computation. *IEEE Transactions on Evolutionary Computation*, 15(5):591–607, 2011b.

Y. P. Chen, T. li Yu, K. Sastry, and D. E. Goldberg. A survey of linkage learning techniques in genetic and evolutionary algorithms. Technical report, Illinois Genetic Algorithms Library, April 2007.

R. Cheng and Y. Jin. A competitive swarm optimizer for large scale optimization. *IEEE transactions on cybernetics*, 45(2):191–204, 2015.

S. Cheng, Y. Shi, and Q. Qin. Dynamical exploitation space reduction in particle swarm optimization for solving large scale problems. In *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pages 1–8. IEEE, 2012.

S. Cheng, Y. Shi, Q. Qin, and R. Bai. Swarm intelligence in big data analytics. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 417–426. Springer, 2013.

S. Cheng, Y. Shi, Q. Qin, T. O. Ting, and R. Bai. Maintaining population diversity in brain storm optimization algorithm. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*, pages 3230–3237. IEEE, 2014.

S. Cheng, Q. Zhang, and Q. Qin. Big data analytics with swarm intelligence. *Industrial Management & Data Systems*, 116(4):646–666, 2016.

R. Chevrier, P. Pellegrini, and J. Rodriguez. Energy saving in railway timetabling: A bi-objective evolutionary approach for computing alternative running times. *Transportation Research Part C: Emerging Technologies*, 37:20–41, 2013.

C. Chira, J. Sedano, J. R. Villar, M. Cámara, and E. Corchado. Urban bicycles renting systems: Modelling and optimization using nature-inspired search methods. *Neurocomputing*, 135:98–106, 2014.

C.-H. Chou and J.-N. Chen. Genetic algorithms: initialization schemes and genes extraction. In *Fuzzy Systems, 2000. FUZZ IEEE 2000. The Ninth IEEE International Conference on*, volume 2, pages 965–968. IEEE, 2000.

Y. Chu, H. Mi, H. Liao, Z. Ji, and Q. Wu. A fast bacterial swarming algorithm for high-dimensional function optimization. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 3135–3140. IEEE, 2008.

M. Clerc. Initialisations for particle swarm optimisation. *Online at http://clerc. maurice. free. fr/pso*, 2008.

C. A. Coello Coello. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. *Computer methods in applied mechanics and engineering*, 191(11):1245–1287, 2002.

C. A. Coello Coello, D. A. Van Veldhuizen, and G. B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, New York, USA, 2002.

B. Colson and P. L. Toint. Optimizing partially separable functions without derivatives. *Optimization methods and software*, 20(4-5):493–508, 2005.

A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to derivative-free optimization*. SIAM, 2009.

P. A. Consoli, L. L. Minku, and X. Yao. Dynamic selection of evolutionary algorithm operators based on online learning and fitness landscape metrics. In *Simulated Evolution and Learning*, pages 359–370. Springer, 2014.

G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1):101–111, 1960.

S. Das and P. N. Suganthan. Differential evolution: A survey of the state-of-the-art. *Evolutionary Computation, IEEE Transactions on*, 15(1):4–31, 2011.

S. Das, A. Konar, and U. K. Chakraborty. Two improved differential evolution schemes for faster global search. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 991–998. ACM, 2005.

D. Dasgupta, G. Hernandez, A. Romero, D. Garrett, A. Kaushal, and J. Simien. On the use of informed initialization and extreme solutions sub-population in multi-objective evolutionary algorithms. In *IEEE Symposium on Computational Intelligence in Miulti-criteria Decision-making, 2009. MCDM'09.*, pages 58–65. IEEE, 2009a.

S. Dasgupta. Learning mixtures of gaussians. In *Foundations of computer science, 1999. 40th annual symposium on*, pages 634–644. IEEE, 1999.

S. Dasgupta, A. Biswas, S. Das, B. K. Panigrahi, and A. Abraham. A micro-bacterial foraging algorithm for high-dimensional optimization. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 785–792. IEEE, 2009b.

Y. Davidor. Epistasis Variance: Suitability of a Representation to Genetic Algorithms. *Complex Systems*, 4(4):369–383, 1990.

V. de Albuquerque Torreão and R. Vimieiro. Effects of population initialization on evolutionary techniques for subgroup discovery in high dimensional datasets. In *2018 7th Brazilian Conference on Intelligent Systems (BRACIS)*, pages 25–30. IEEE, 2018.

V. V. de Melo and A. C. Botazzo Delbem. Investigating smart sampling as a population initialization method for differential evolution in continuous problems. *Information Sciences*, 193:36–53, 2012a.

V. V. de Melo and A. C. Botazzo Delbem. Investigating smart sampling as a population initialization method for differential evolution in continuous problems. *Information Sciences*, 193:36–53, 2012b.

M. A. M. de Oca, D. Aydın, and T. Stützle. An incremental particle swarm for large-scale continuous optimization problems: an example of tuning-in-the-loop (re) design of optimization algorithms. *Soft Computing*, 15(11):2233–2255, 2011.

G. del Puente. Online social networks influence maximization, bio-inspired approaches. Master's thesis, Scuola Politecnica e delle Scienze di Base Corso di Laurea Magistrale in Ingegneria Informatica, 2017.

J. Derrac, S. García, D. Molina, and F. Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18, 2011.

J. Derrac, I. Triguero, S. García, and F. Herrera. Integrating instance selection, instance weighting, and feature weighting for nearest neighbor classifiers by coevolutionary algorithms. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(5):1383–1397, 2012.

J. Dick and F. Pillichshammer. On the mean square weighted l2 discrepancy of randomized digital (t, m, s)-nets over z2. *Acta Arith*, 117(371-403):533–560, 2005.

E. D. Dolan, J. J. More, and T. S. Munson. Benchmarking optimization software with COPS 3.0. Technical report, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois 60439, 2004.

N. Dong, C.-H. Wu, W.-H. Ip, Z.-Q. Chen, C.-Y. Chan, and K.-L. Yung. An opposition-based chaotic ga/pso hybrid algorithm and its application in circle detection. *Computers & Mathematics with Applications*, 64(6):1886–1902, 2012.

W. Dong and X. Yao. Unified eigen analysis on multivariate gaussian based estimation of distribution algorithms. *Information Sciences*, 178(15):3000–3023, 2008.

W. Dong, T. Chen, P. Tino, and X. Yao. Scaling up estimation of distribution algorithms for continuous optimization. *IEEE Transactions on Evolutionary Computation*, 17(6): 797–822, 2013.

M. Dorigo, V. Maniezzo, and A. Colorni. The Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics*, 26(1):29–41, 1996.

C. Dutang and D. Wuertz. A note on random number generation. *Overview of Random Generation Algoritms*, pages 1–29, 2009.

A. Ebert and P. Kritzer. Constructing lattice points for numerical integration by a reduced fast successive coordinate search algorithm. *arXiv preprint arXiv:1804.01765*, 2018.

C. Echegoyen, Q. Zhang, A. Mendiburu, R. Santana, and J. A. Lozano. On the limits of effectiveness in estimation of distribution algorithms. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 1573–1580. IEEE, 2011.

M. El-Abd. Opposition-based artificial bee colony algorithm. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 109–116. ACM, 2011.

M. El-Abd. Generalized opposition-based artificial bee colony algorithm. In *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pages 1–4. IEEE, 2012.

M. El-Abd. Hybrid cooperative co-evolution for large scale optimization. In *IEEE Symposium on Swarm Intelligence (SIS'14)*, pages 1–6. IEEE, 2014.

M. Ergezer and I. Sikder. Survey of oppositional algorithms. In *Computer and Information Technology (ICCIT), 2011 14th International Conference on*, pages 623–628. IEEE, 2011.

M. Ergezer, D. Simon, and D. Du. Oppositional biogeography-based optimization. In *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, pages 1009–1014. IEEE, 2009.

S. Ergün and S. Özoguz. Truly random number generators based on non-autonomous continuous-time chaos. *international journal of circuit theory and applications*, 38(1): 1–24, 2010.

L. Eshelman. The CHC adaptive search algorithm. foundations of genetic algorithms, g. rawlins, ed, 1991.

M. Essaid, L. Idoumghar, J. Lepagnot, M. Brévilliers, and D. Fodorean. A hybrid optimization algorithm for electric motor design. In *International Conference on Computational Science*, pages 501–517. Springer, 2018.

J. Fan, J. Wang, and M. Han. Cooperative coevolution for large-scale optimization based on kernel fuzzy clustering and variable trust region methods. *IEEE Transactions on Fuzzy Systems*, 22(4):829–839, 2014.

K.-T. Fang and D. K. Lin. Uniform experimental designs and their applications in industry. *Handbook of Statistics*, 22:131–170, 2003.

Á. Fialho, M. Schoenauer, and M. Sebag. Analysis of adaptive operator selection techniques on the royal road and long k-path problems. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 779–786. ACM, 2009.

Á. Fialho, L. Da Costa, M. Schoenauer, and M. Sebag. Analyzing bandit-based adaptive operator selection mechanisms. *Annals of Mathematics and Artificial Intelligence*, 60 (1):25–64, 2010a. ISSN 1573-7470. URL http://dx.doi.org/10.1007/s10472-010-9213-y.

Á. Fialho, M. Schoenauer, and M. Sebag. Toward comparison-based adaptive operator selection. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 767–774. ACM, 2010b.

L. Fogel, A. Owens, and M. Walsh. *Artificial intelligence through simulated evolution.* Wiley, Chichester, WS, UK, 1966.

A. A. Freitas. A review of evolutionary algorithms for data mining. In *Data Mining and Knowledge Discovery Handbook*, pages 371–400. Springer, 2009.

J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

Z. Fuxing, Z. Tao, and W. Rui. Gbest-guided covariance matrix adaptation evolution strategy for large scale global optimization. In *International Conference on Intelligent Computing*, pages 3–13. Springer, 2017.

R. D. Gaina, S. M. Lucas, and D. Pérez-Liébana. Population seeding techniques for rolling horizon evolution in general video game playing. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 1956–1963. IEEE, 2017.

R. Gamperle, S. Muller, and P. Koumoutsakos. A parameter study for differential evolution. In *Proc. of WSEAS International Conference on Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*, pages 293–298, 2002.

W.-f. Gao and S.-y. Liu. A modified artificial bee colony algorithm. *Computers & Operations Research*, 39(3):687–697, 2012.

W.-f. Gao, S.-y. Liu, and L.-l. Huang. Particle swarm optimization with chaotic opposition-based population initialization and stochastic search technique. *Communications in Nonlinear Science and Numerical Simulation*, 17(11):4316–4327, 2012.

Y. Gao and Y.-J. Wang. A memetic differential evolutionary algorithm for high dimensional functions' optimization. In *Natural Computation, 2007. ICNC 2007. Third International Conference on*, volume 4, pages 188–192. IEEE, 2007.

S. García and F. Herrera. Evolutionary undersampling for classification with imbalanced datasets: Proposals and taxonomy. *Evolutionary computation*, 17(3):275–306, 2009.

S. García, J. R. Cano, and F. Herrera. A memetic algorithm for evolutionary prototype selection: A scaling up approach. *Pattern Recognition*, 41(8):2693–2709, 2008.

S. García, A. Fernández, J. Luengo, and F. Herrera. Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences*, 180(10):2044–2064, 2010.

M. García-Arnau, D. Manrique, J. Rios, and A. Rodríguez-Patón. Initialization method for grammar-guided genetic programming. *Knowledge-Based Systems*, 20(2):127–133, 2007.

C. García-Martínez and M. Lozano. Continuous variable neighbourhood search algorithm based on evolutionary metaheuristic components: a scalability test. In *Intelligent Systems Design and Applications, 2009. ISDA'09. Ninth International Conference on*, pages 1074–1079. IEEE, 2009.

J. García-Nieto and E. Alba. Restart particle swarm optimization with velocity modulation: a scalability test. *Soft Computing*, 15(11):2221–2232, 2011.

N. García-Pedrajas. Evolutionary computation for training set selection. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(6):512–523, 2011.

N. García-Pedrajas and A. de Haro-García. Scaling up data mining algorithms: review and taxonomy. *Progress in Artificial Intelligence*, 1(1):71–87, 2012.

N. García-Pedrajas, C. Hervás-Martínez, and J. Muñoz-Pérez. COVNET: a cooperative coevolutionary model for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, 14(3):575–596, 2003.

N. García-Pedrajas, C. Hervás-Martínez, and D. Ortiz-Boyer. Cooperative coevolution of artificial neural network ensembles for pattern classification. *IEEE transactions on evolutionary computation*, 9(3):271–302, 2005.

N. García-Pedrajas, J. A. R. Del Castillo, and D. Ortiz-Boyer. A cooperative coevolutionary algorithm for instance selection for instance-based learning. *Machine Learning*, 78 (3):381–420, 2010.

N. GarcíA-Pedrajas, A. De Haro-GarcíA, and J. PéRez-RodríGuez. A scalable approach to simultaneous evolutionary instance and feature selection. *Information Sciences*, 228: 150–174, 2013.

V. Gardeux, M. G. Omran, R. Chelouah, P. Siarry, and F. Glover. Adaptive pattern search for large-scale optimization. *Applied Intelligence*, pages 1–12, 2017.

M. Gardner. The bells: versatile numbers that can count partitions of a set, primes and even rhymes. *Scientific American*, 238:24–26, 1978.

H. Ge, L. Sun, and X. Yang. Adaptive hybrid differential evolution with circular sliding window for large scale optimization. In *Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), 2016 12th International Conference on*, pages 87–94. IEEE, 2016.

H. Ge, L. Sun, G. Tan, Z. Chen, and C. P. Chen. Cooperative hierarchical pso with two stage variable interaction reconstruction for large scale optimization. *IEEE Transactions on Cybernetics*, 2017.

N. Geroliminis, K. Kepaptsoglou, and M. G. Karlaftis. A hybrid hypercube–genetic algorithm approach for deploying many emergency response mobile units in an urban network. *European Journal of Operational Research*, 210(2):287–300, 2011.

J. Gittins, K. Glazebrook, and R. Weber. *Multi-armed bandit allocation indices*. John Wiley & Sons, 2011.

D. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3(5):493–530, 1989.

D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

D. E. Goldberg, K. Deb, H. Kargupta, and G. Harik. Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In S. Forrest, editor, *Proc. of International Conference on Genetic Algorithms*, pages 56–64, San Francisco, CA, 1993. Morgan Kaufmann.

M. Gong, L. Jiao, F. Liu, and W. Ma. Immune algorithm with orthogonal design based initialization, cloning, and selection for global optimization. *Knowledge and information systems*, 25(3):523–549, 2010.

W. Gong, Á. Fialho, Z. Cai, and H. Li. Adaptive strategy selection in differential evolution for numerical optimization: an empirical study. *Information Sciences*, 181(24):5364–5386, 2011.

A. Gopakumar and L. Jacob. Localization in wireless sensor networks using particle swarm optimization. In *Conference on Wireless, Mobile and Multimedia Networks*, page 227 – 230. IET, 2008.

A. Griewank and P. L. Toint. Local convergence analysis for partitioned quasi-newton updates. *Numerische Mathematik*, 39(3):429–448, 1982.

J. L. Guerrero, A. Berlanga, and J. M. Molina. Initialization procedures for multiobjective evolutionary approaches to the segmentation issue. In *Hybrid Artificial Intelligent Systems*, pages 452–463. Springer, 2012.

S. Guru, S. Halgamuge, and S. Fernando. Particle swarm optimisers for cluster formation in wireless sensor networks. In *Intelligent Sensors, Sensor Networks and Information Processing Conference, 2005. Proceedings of the 2005 International Conference on*, pages 319–324. IEEE, 2005.

A. Gutiérrez, M. Lanza, I. Barriuso, L. Valle, M. Domingo, J. Perez, and J. Basterrechea. Comparison of different pso initialization techniques for high dimensional search space problems: A test with fss and antenna arrays. In *Antennas and Propagation (EUCAP), Proceedings of the 5th European Conference on*, pages 965–969. IEEE, 2011.

M. Haahr. Random. org: True random number service. *School of Computer Science and Statistics, Trinity College, Dublin, Ireland. Website (http://www. random. org). Accessed*, 10, 2010.

A. A. Hadi, A. W. Mohamed, and K. M. Jambi. Lshade-spa memetic framework for solving large-scale optimization problems. *Complex & Intelligent Systems*, 5(1):25–40, 2019.

J. H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2(1):84–90, 1960.

N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

N. Hansen, S. Finck, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. Technical Report RR-6829, INRIA, 2010.

G. Harik, F. Lobo, and D. Goldberg. The compact genetic algorithm. *IEEE Transactions on Evolutionary Computation*, 3(4):287–297, November 1999.

G. R. Harik. *Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms*. PhD thesis, University of Michigan, Ann Arbor, MI, 1997.

S. He, G. Jia, Z. Zhu, D. A. Tennant, Q. Huang, K. Tang, J. Liu, M. Musolesi, J. K. Heath, and X. Yao. Cooperative co-evolutionary module identification with application to cancer disease module discovery. *IEEE Transactions on Evolutionary Computation*, 20(6):874–891, 2016.

S. Helwig and R. Wanka. Theoretical analysis of initial particle swarm behavior. In *Parallel Problem Solving from Nature–PPSN X*, pages 889–898. Springer, 2008.

F. Herrera and M. Lozano. Two-loop real-coded genetic algorithms with adaptive control of mutation step sizes. *Applied Intelligence*, 13(3):187–204, 2000.

M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49:409–436, 1952.

F. Hickernell. A generalized discrepancy and quadrature error bound. *Mathematics of Computation of the American Mathematical Society*, 67(221):299–322, 1998.

J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, USA, 1975.

J.-H. Hong and S.-B. Cho. Efficient huge-scale feature selection with speciated genetic algorithm. *Pattern Recognition Letters*, 27(2):143–150, 2006.

T.-P. Hong and G.-N. Shiu. Allocating multiple base stations under general power consumption by the particle swarm optimization. In *Swarm Intelligence Symposium, 2007. SIS 2007. IEEE*, pages 23–28. IEEE, 2007.

S.-T. Hsieh, T.-Y. Sun, C.-C. Liu, and S.-J. Tsai. Solving large scale global optimization using improved particle swarm optimizer. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 1777–1784. IEEE, 2008.

Q. Huang, T. White, G. Jia, M. Musolesi, N. Turan, K. Tang, S. He, J. K. Heath, and X. Yao. Community detection using cooperative co-evolutionary differential evolution. In *International Conference on Parallel Problem Solving from Nature*, pages 235–244. Springer, 2012.

M. A. Iqbal, N. K. Khan, M. A. Jaffar, M. Ramzan, and A. R. Baig. Opposition based genetic algorithm with cauchy mutation for function optimization. In *Information Science and Applications (ICISA), 2010 International Conference on*, pages 1–7. IEEE, 2010.

H. Ishibuchi and S. Namba. Evolutionary multiobjective knowledge extraction for high-dimensional pattern classification problems. In *PPSN*, pages 1123–1132. Springer, 2004.

H. Jabeen, Z. Jalil, and A. R. Baig. Opposition based initialization in particle swarm optimization (o-pso). In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, pages 2047–2052. ACM, 2009.

T. Jansen, I. Wegener, K. Tinnefeld, and S. Droste. A new framework for the valuation of algorithms for black-box-optimization. Technical report, Collaborative Research Center 531, "Computational Intelligence", University of Dortmund, 2001.

Y. Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, 9(1):3–12, 2005.

Y. Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, 1(2):61–70, 2011.

D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

J.-C. Jong and P. Schonfeld. An evolutionary model for simultaneously optimizing three-dimensional highway alignments. *Transportation Research Part B: Methodological*, 37 (2):107–128, 2003.

B. Jun and P. Kocher. The intel random number generator. *Cryptography Research Inc. white paper*, 1999.

A. Kabán, J. Bootkrajang, and R. J. Durrant. Toward large-scale continuous eda: a random matrix theory perspective. *Evolutionary computation*, 24(2):255–291, 2016.

H. Kargupta. The performance of the gene expression messy genetic algorithm on real test functions. In *IEEE Congress on Evolutionary Computation (CEC'96)*, pages 631–636, may 1996.

S. Karol and V. Mangat. Evaluation of text document clustering approach based on particle swarm optimization. *Open Computer Science*, 3(2):69–90, 2013.

S. Kazemzadeh Azad. Seeding the initial population with feasible solutions in metaheuristic optimization of steel trusses. *Engineering Optimization*, 50(1):89–105, 2018.

B. Kazimipour, X. Li, and A. K. Qin. Initialization methods for large scale global optimization. In *IEEE Congress on Evolutionary Computation (CEC'13)*, pages 2750–2757. IEEE, 2013.

B. Kazimipour, X. Li, and A. K. Qin. Effects of population initialization in differential evolution for large scale optimization. In *IEEE Congress on Evolutionary Computation (CEC'14)*, pages 2404–2411. IEEE, 2014a.

B. Kazimipour, X. Li, and A. K. Qin. A review of population initialization techniques for evolutionary algorithms. In *IEEE Congress on Evolutionary Computation (CEC'14)*, pages 2585–2592. IEEE, 2014b.

B. Kazimipour, X. Li, and A. K. Qin. Why advanced population initialization techniques perform poorly in high dimension? In *Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'14)*, pages 479–490. Springer, Springer, 2014c.

B. Kazimipour, M. N. Omidvar, X. Li, and A. K. Qin. A novel hybridization of opposition-based learning and cooperative co-evolutionary for large-scale optimization. In *IEEE Congress on Evolutionary Computation (CEC'14)*, pages 2833–2840. IEEE, 2014d.

B. Kazimipour, M. N. Omidvar, X. Li, and A. Qin. A sensitivity analysis of contribution-based cooperative co-evolutionary algorithms. In *IEEE Congress on Evolutionary Computation (CEC'15)*, pages 417–424. IEEE, 2015.

B. Kazimipour, M. N. Omidvar, A. Qin, X. Li, and X. Yao. Bandit-based cooperative coevolution for tackling contribution imbalance in large-scale optimization problems. *Applied Soft Computing*, 76:265–281, 2019.

J. Kennedy. Particle swarm optimization. In *Encyclopedia of machine learning*, pages 760–766. Springer, 2011.

J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proc. of IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

R. A. Khanum and M. A. Jan. Centroid-based initialized jade for global optimization. In *Computer Science and Electronic Engineering Conference (CEEC), 2011 3rd*, pages 115–120. IEEE, 2011.

M. Kim, R. I. B. McKay, D.-K. Kim, and X. H. Nguyen. Evolutionary operator self-adaptation with diverse operators. In *Genetic Programming*, pages 230–241. Springer, 2012.

S. Kimura and K. Matsumura. Genetic algorithms using low-discrepancy sequences. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1341–1346. ACM, 2005.

S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, et al. Optimization by simulated annealing. *Science Magazine*, 220(4598):671–680, 1983.

K. C. Kiwiel. Convergence and efficiency of subgradient methods for quasiconvex minimization. *Mathematical programming*, 90(1):1–25, 2001.

W. S. Klug, M. R. Cummings, C. Spencer, C. A. Spencer, and M. A. Palladino. *Concepts of Genetics*. Pearson, 9 edition, 2008. Epistasis.

B.-I. Koh, J. A. Reinbolt, A. D. George, R. T. Haftka, and B. J. Fregly. Limitations of parallel global optimization for large-scale human movement problems. *Medical engineering & physics*, 31(5):515–521, 2009.

A. V. Kononova, D. B. Ingham, and M. Pourkashanian. Simple scheduled memetic algorithm for inverse problems in higher dimensions: Application to chemical kinetics. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 3905–3912. IEEE, 2008.

V. Kuleshov and D. Precup. Algorithms for multi-armed bandit problems. *arXiv preprint arXiv:1402.6028*, 2014.

R. V. Kulkarni and G. K. Venayagamoorthy. Particle swarm optimization in wireless-sensor networks: A brief survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 41(2):262–267, 2011.

R. V. Kulkarni, A. Forster, and G. K. Venayagamoorthy. Computational intelligence in wireless sensor networks: A survey. *IEEE communications surveys & tutorials*, 13(1): 68–96, 2011.

R. Kumar, S. Narula, and R. Kumar. A population initialization method by memetic algorithm. *International Journal*, 3(4), 2013.

M. Lahanas. Application of multiobjective evolutionary optimization algorithms in medicine. *Applications of Multi-Objective Evolutionary Algorithms*, pages 365–391, 2004.

P. Larrañaga and J. Lozano. *Estimation of Distribution Algorithms: A new tool for evolutionary computation*. Kluwer Academic Pub, 2002.

P. Larrañaga and J. A. Lozano. *Estimation of distribution algorithms: A new tool for evolutionary computation*, volume 2. Springer Science & Business Media, 2001.

A. LaTorre, S. Muelas, and J.-M. Peña. A MOS-based dynamic memetic differential evolution algorithm for continuous optimization: a scalability test. *Soft Computing*, 15 (11):2187–2199, 2011.

A. LaTorre, S. Muelas, and J.-M. Peña. Multiple offspring sampling in large scale global optimization. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2012.

A. LaTorre, S. Muelas, and J.-M. Peña. Large scale global optimization: Experimental results with MOS-based hybrid algorithms. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 2742–2749. IEEE, 2013.

A. LaTorre, S. Muelas, and J.-M. Peña. A comprehensive comparison of large scale global optimizers. *Information Sciences*, 316:517–549, 2015.

A. LaTorre de la Fuente. *A framework for hybrid dynamic evolutionary algorithms: multiple offspring sampling (MOS)*. PhD thesis, Informatica, 2009.

M. N. Le, Y. S. Ong, S. Menzel, Y. Jin, and B. Sendhoff. Evolution by adapting surrogates. *Evolutionary computation*, 21(2):313–340, 2013.

P. L'Ecuyer and R. Simard. Testu01: Ac library for empirical testing of random number generators. *ACM Transactions on Mathematical Software (TOMS)*, 33(4):22, 2007.

M. Lescrenier. Partially separable optimization and parallel computing. *Annals of Operations Research*, 14(1):213–224, 1988.

Y.-W. Leung and Y. Wang. An orthogonal genetic algorithm with quantization for global numerical optimization. *Evolutionary Computation, IEEE Transactions on*, 5(1):41–53, 2001.

K. Li, A. Fialho, S. Kwong, and Q. Zhang. Adaptive operator selection with bandits for a multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 18(1):114–130, 2014.

X. Li. Decomposition and cooperative coevolution techniques for large scale global optimization. In *The 2014 Conference on Genetic and Evolutionary Computation (GECCO'14)*, pages 819–838, 2014.

X. Li and X. Yao. Tackling high dimensional nonseparable optimization problems by cooperatively coevolving particle swarms. In *Proc. of IEEE Congress on Evolutionary Computation*, pages 1546–1553, 2009.

X. Li and X. Yao. Cooperatively coevolving particle swarms for large scale optimization. *IEEE Transactions on Evolutionary Computation*, 16(2):210–224, April 2012.

X. Li, N. Xiao, C. Claramunt, and H. Lin. Initialization strategies to enhancing the performance of genetic algorithms for the¡ i¿ p¡/i¿-median problem. *Computers & Industrial Engineering*, 61(4):1024–1034, 2011.

X. Li, K. Tang, M. N. Omidvar, Z. Yang, and K. Qin. Benchmark functions for the CEC'2013 special session and competition on large-scale global optimization. Technical report, RMIT University, Melbourne, Australia, 2013a. URL http://goanna.cs.rmit.edu.au/~xiaodong/cec13-lsgo.

Y. Li, G. Wang, H. Chen, L. Shi, and L. Qin. An ant colony optimization based dimension reduction method for high-dimensional datasets. *Journal of Bionic Engineering*, 10(2): 231–241, 2013b.

J.-J. Liang and P. N. Suganthan. Dynamic multi-swarm particle swarm optimizer with local search. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 1, pages 522–528. Ieee, 2005. DMS-PSO.

T. Liao, M. A. Montes de Oca, D. Aydin, T. Stützle, and M. Dorigo. An incremental ant colony algorithm with local search for continuous optimization. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 125–132. ACM, 2011.

D. Lim, Y. Jin, Y.-S. Ong, and B. Sendhoff. Generalizing surrogate-assisted evolutionary computation. *IEEE Transactions on Evolutionary Computation*, 14(3):329–355, 2010.

L. Lin, M. Gen, and Y. Liang. A hybrid ea for high-dimensional subspace clustering problem. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*, pages 2855–2860. IEEE, 2014.

Z. Lin and L. Wang. A new opposition-based compact genetic algorithm with fluctuation. *J. Comput. Inf. Syst*, 6(3):897–904, 2010.

B. Liu, L. Wang, Y.-H. Jin, F. Tang, and D.-X. Huang. Improved particle swarm optimization combined with chaos. *Chaos, Solitons & Fractals*, 25(5):1261–1271, 2005.

J. Liu and K. Tang. Scaling up covariance matrix adaptation evolution strategy using cooperative coevolution. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 350–357. Springer, 2013.

Y. Liu, X. Yao, Q. Zhao, and T. Higuchi. Scaling up fast evolutionary programming with cooperative coevolution. In *IEEE Congress on Evolutionary Computation (CEC'01)*, volume 2, pages 1101–1108. IEEE, 2001.

S. Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982.

M. Lozano and C. García-Martínez. An evolutionary ils-perturbation technique. In *Hybrid Metaheuristics*, pages 1–15. Springer, 2008.

Y. Lu, S. Wang, S. Li, and C. Zhou. Text clustering via particle swarm optimization. In *Swarm Intelligence Symposium, 2009. SIS'09. IEEE*, pages 45–51. IEEE, 2009.

Y. Lu, S. Wang, S. Li, and C. Zhou. Particle swarm optimizer for variable weighting in clustering high-dimensional data. *Machine learning*, 82(1):43–70, 2011.

C. B. Lucasius and G. Kateman. Genetic algorithms for large-scale optimization in chemometrics: An application. *TrAC Trends in Analytical Chemistry*, 10(8):254 – 261, 1991.

Z. Ma and G. A. Vandenbosch. Impact of random number generators on the performance of particle swarm optimization in antenna design. In *Antennas and Propagation (EUCAP), 2012 6th European Conference on*, pages 925–929. IEEE, 2012.

H. Maaranen, K. Miettinen, and M. M. Mäkelä. Quasi-random initial population for genetic algorithms. *Computers & Mathematics with Applications*, 47(12):1885–1895, 2004.

H. Maaranen, K. Miettinen, and A. Penttinen. On initial populations of a genetic algorithm for continuous optimization problems. *Journal of Global Optimization*, 37(3): 405–436, 2007.

J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, pages 281–297. Oakland, CA, USA, 1967.

S. Mahdavi, M. E. Shiri, and S. Rahnamayan. Cooperative co-evolution with a new decomposition method for large-scale optimization. In *IEEE Congress on Evolutionary Computation (CEC'14)*, pages 1285–1292. IEEE, 2014a.

S. Mahdavi, M. E. Shiri, and S. Rahnamayan. Metaheuristics in large-scale global continues optimization: A survey. *Information Sciences*, 2014b.

S. Mahdavi, S. Rahnamayan, and K. Deb. Center-based initialization of cooperative co-evolutionary algorithm for large-scale optimization. In *Evolutionary Computation (CEC), 2016 IEEE Congress on*, pages 3557–3565. IEEE, 2016a.

S. Mahdavi, S. Rahnamayan, and M. E. Shiri. Incremental cooperative coevolution for large-scale global optimization. *Soft Computing*, pages 1–20, 2016b.

S. Mahdavi, S. Rahnamayan, and M. E. Shiri. Multilevel framework for large-scale global optimization. *Soft Computing*, pages 1–30, 2016c.

S. Mahdavi, S. Rahnamayan, and M. E. Shiri. Cooperative co-evolution with sensitivity analysis-based budget assignment strategy for large-scale global optimization. *Applied Intelligence*, pages 1–26, 2017.

X.-f. Mai and L. Li. Bacterial foraging optimization algorithm based on opposition-based learning. *Energy Procedia*, 13:5726–5732, 2011.

R. Mallipeddi, P. N. Suganthan, Q.-K. Pan, and M. F. Tasgetiren. Differential evolution algorithm with ensemble of parameters and mutation strategies. *Applied Soft Computing*, 11(2):1679–1696, 2011.

T. W. Manikas, K. Ashenayi, and R. L. Wainwright. Genetic algorithms for autonomous robot navigation. *IEEE Instrumentation & Measurement Magazine*, 10(6), 2007.

E. Marchiori and A. Steenbeek. An evolutionary algorithm for large scale set covering problems with application to airline crew scheduling. In *Real-World Applications of Evolutionary Computing*, pages 370–384. Springer, 2000.

G. Marsaglia and A. Zaman. The kiss generator. Technical report, Tech. rep., Department of Statistics, University of Florida, 1993.

M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.

M. J. Meena and S. S. Ibrahim. Statistical and evolutionary feature selection techniques parallelized using mapreduce programming model. In *Techniques and Environments for Big Data Analysis*, pages 159–180. Springer, 2016.

S. Mehta. Memetic algorithm with constrained local search for large-scale global optimization. *Journal of Intelligent Systems*, 26(2):287–300, 2017.

Y. Mei, K. Tang, and X. Yao. Decomposition-based memetic algorithm for multiobjective capacitated arc routing problem. *IEEE Transactions on Evolutionary Computation*, 15 (2):151–165, 2011.

Y. Mei, X. Li, and X. Yao. Cooperative coevolution with route distance grouping for large-scale capacitated arc routing problems. *IEEE Transactions on Evolutionary Computation*, 18(3):435–449, 2014a.

Y. Mei, X. Li, and X. Yao. Variable neighborhood decomposition for large scale capacitated arc routing problem. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*, pages 1313–1320. IEEE, 2014b.

Y. Mei, M. N. Omidvar, X. Li, and X. Yao. A competitive divide-and-conquer algorithm for unconstrained large-scale black-box optimization. *ACM Transactions on Mathematical Software (TOMS)*, 42(2):13, 2016. Global Differential Grouping (GDG).

E. Mezura-Montes and C. A. Coello Coello. Constraint-handling in nature-inspired numerical optimization: past, present and future. *Swarm and Evolutionary Computation*, 1(4):173–194, 2011.

Z. Michalewicz and D. B. Fogel. *How to solve it: modern heuristics*. Springer Science & Business Media, 2013.

K. Miettinen. *Nonlinear Multiobjective Optimization*. Norwell, MA: Kluwer, 1999.

X. Mingming, Z. Jun, C. Kaiquan, C. Xianbin, and T. Ke. Cooperative co-evolution with weighted random grouping for large-scale crossing waypoints locating in air route network. In *Tools with Artificial Intelligence (ICTAI), 2011 23rd IEEE International Conference on*, pages 215–222. IEEE, 2011.

P. Mohapatra, K. N. Das, and S. Roy. A modified competitive swarm optimizer for large scale optimization problems. *Applied Soft Computing*, 59:340–362, 2017.

D. Molina, M. Lozano, and F. Herrera. Memetic algorithm with local search chaining for large scale continuous optimization problems. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 830–837. IEEE, 2009.

D. Molina, M. Lozano, and F. Herrera. MA-SW-Chains: Memetic algorithm based on local search chains for large scale continuous global optimization. In *IEEE Congress on Evolutionary Computation (CEC'10)*, pages 3153–3160. IEEE, july 2010.

D. Molina, M. Lozano, A. M. Sánchez, and F. Herrera. Memetic algorithms based on local search chains for large scale continuous optimisation problems: Ma-ssw-chains. *Soft Computing*, 15(11):2201–2220, 2011.

D. Molina, A. R. Nesterenko, and A. LaTorre. Comparing large-scale global optimization competition winners in a real-world problem. In *2019 IEEE Congress on Evolutionary Computation (CEC)*, pages 359–365. IEEE, 2019.

W. J. Morokoff and R. E. Caflisch. Quasi-random sequences and their discrepancies. *SIAM Journal on Scientific Computing*, 15(6):1251–1279, 1994.

M. D. Morris. Factorial sampling plans for preliminary computational experiments. *Technometrics*, 33(2):161–174, 1991.

R. W. Morrison. Dispersion-based population initialization. In *Genetic and Evolutionary Computation—GECCO 2003*, pages 1210–1221. Springer, 2003.

H. Muhlenbein and T. Mahnig. Convergence theory and applications of the factorized distribution algorithm. *CIT. Journal of computing and information technology*, 7(1): 19–32, 1999.

H. Mühlenbein and G. Paass. From recombination of genes to the estimation of distributions i. binary parameters. In *Proc. of International Conference on Parallel Problem Solving from Nature*, pages 178–187, London, UK, 1996. Springer-Verlag.

H. Mühlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimizer. *Parallel computing*, 17(6-7):619–632, 1991.

M. Munetomo and D. Goldberg. A genetic algorithm using linkage identification by nonlinearity check. In *Proc. of IEEE International Conference on Systems, Man, and Cybernetics*, volume 1, pages 595–600, 1999.

A. C. Nearchou. Adaptive navigation of autonomous vehicles using evolutionary algorithms. *Artificial Intelligence in Engineering*, 13(2):159–173, 1999.

G. L. Nemhauser and L. A. Wolsey. Integer programming and combinatorial optimization. *Wiley, Chichester. GL Nemhauser, MWP Savelsbergh, GS Sigismondi (1992). Constraint Classification for Mixed Integer Programming Formulations. COAL Bulletin*, 20:8–12, 1988.

M. Olguin-Carbajal, E. Alba, and J. Arellano-Verdejo. Micro-differential evolution with local search for high dimensional problems. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 48–54. IEEE, 2013a.

M. Olguin-Carbajal, J. C. Herrera-Lozada, J. Arellano-Verdejo, R. Barron-Fernandez, and H. Taud. Micro differential evolution performance empirical study for high dimensional optimization problems. In *International Conference on Large-Scale Scientific Computing*, pages 281–288. Springer, 2013b.

M. N. Omidvar. *Cooperative Co-evolutionary Algorithms for Large-Scale Optimization.* phdthesis, School of Computer Science and Information Technology College of Science Engineering and Health RMIT University, Dec. 2015.

M. N. Omidvar, X. Li, Z. Yang, and X. Yao. Cooperative co-evolution for large scale optimization through more frequent random grouping. In *IEEE Congress on Evolutionary Computation (CEC'10)*, pages 1754–1761. IEEE, 2010a.

M. N. Omidvar, X. Li, and X. Yao. Cooperative co-evolution with delta grouping for large scale non-separable function optimization. In *IEEE Congress on Evolutionary Computation (CEC'10)*, pages 1762–1769. IEEE, 2010b.

M. N. Omidvar, X. Li, and X. Yao. Smart use of computational resources based on contribution for cooperative co-evolutionary algorithms. In *Proc. of Genetic and Evolutionary Computation Conference*, pages 1115–1122. ACM, ACM, 2011.

M. N. Omidvar, X. Li, Y. Mei, and X. Yao. Cooperative co-evolution with differential grouping for large scale optimization. *IEEE Transactions on Evolutionary Computation*, 18(3):378–393, June 2014a. ISSN 1089-778X. doi: 10.1109/TEVC.2013.2281543.

M. N. Omidvar, Y. Mei, and X. Li. Effective decomposition of large-scale separable continuous functions for cooperative co-evolutionary algorithms. In *IEEE Congress on Evolutionary Computation (CEC'14)*, pages 1305–1312. IEEE, 2014b.

M. N. Omidvar, X. Li, and K. Tang. Designing benchmark problems for large-scale continuous optimization. *Information Sciences*, 316:419 – 436, 2015. doi: http://dx.doi.org/10.1016/j.ins.2014.12.062.

M. N. Omidvar, B. Kazimipour, X. Li, and X. Yao. CBCC3 - A contribution-based cooperative co-evolutionary algorithm with improved exploration/exploitation balance. In *IEEE Congress on Evolutionary Computation (CEC'16)*. IEEE, 2016.

M. N. Omidvar, M. Yang, Y. Mei, X. Li, and X. Yao. DG2: a faster and more accurate differential grouping for large-scale black-box optimization. *IEEE Transactions on Evolutionary Computation*, 2017.

Y.-S. Ong, P. B. Nair, and K. Y. Lum. Max-min surrogate-assisted evolutionary algorithm for robust design. *IEEE Transactions on Evolutionary Computation*, 10(4):392–404, 2006. local surrogate.

T. Osvald et al. Kalibrace modelů stochastické volatility pomocí kvazi-evolučních algoritmů. 2017.

A. B. Owen. *Randomly permuted (t, m, s)-nets and (t, s)-sequences*. Springer, 1995.

S. K. Pal, V. Talwar, and P. Mitra. Web mining in soft computing framework: relevance, state of the art and future directions. *IEEE Transactions on Neural Networks*, 13(5): 1163–1177, 2002.

F. Panneton, P. L'ecuyer, and M. Matsumoto. Improved long-period generators based on linear recurrences modulo 2. *ACM Transactions on Mathematical Software (TOMS)*, 32(1):1–16, 2006.

M. Pant, T. Radha, and V. P. Singh. Particle swarm optimization: Experimenting the distributions of random numbers. In *IICAI*, pages 412–420, 2007.

M. Pant, M. Ali, and V. Singh. Differential evolution using quadratic interpolation for initializing the population. In *Advance Computing Conference, 2009. IACC 2009. IEEE International*, pages 375–380. IEEE, 2009a.

M. Pant, R. Thangaraj, and A. Abraham. Particle swarm optimization: performance tuning and empirical analysis. In *Foundations of Computational Intelligence Volume 3*, pages 101–128. Springer, 2009b.

S. K. Park and K. W. Miller. Random number generators: good ones are hard to find. *Communications of the ACM*, 31(10):1192–1201, 1988.

S.-Y. Park and J.-J. Lee. Stochastic opposition-based learning using a beta distribution in differential evolution. *IEEE transactions on cybernetics*, 46(10):2184–2194, 2016.

K. Parsopoulos and M. Vrahatis. Initializing the particle swarm optimizer using the nonlinear simplex method. *Advances in intelligent systems, fuzzy systems, evolutionary computation*, 216, 2002.

K. E. Parsopoulos. Cooperative micro-differential evolution for high-dimensional problems. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 531–538. ACM, 2009.

K. M. Passino. Biomimicry of bacterial foraging for distributed optimization and control. *IEEE control systems*, 22(3):52–67, 2002.

S. Pattnaik, S. Mohan, and V. Tom. Urban bus transit route network design using genetic algorithm. *Journal of transportation engineering*, 124(4):368–375, 1998.

M. Pelikan. Probabilistic model-building genetic algorithms. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, pages 913–940. ACM, 2011.

M. Pelikan and D. E. Goldberg. BOA: The Bayesian Optimization Algorithm. In *Proc. of Genetic and Evolutionary Computation Conference*, pages 525–532. Morgan Kaufmann, 1999. BOA.

M. Pelikan, M. Pelikan, D. E. Goldberg, and D. E. Goldberg. Escaping hierarchical traps with competent genetic algorithms. In *Proc. of Genetic and Evolutionary Computation Conference*, pages 511–518. Morgan Kaufmann, 2001. hBOA.

M. Pelikan, D. E. Goldberg, and F. G. Lobo. A survey of optimization by building and using probabilistic models. *Comp. Opt. and Appl.*, 21(1):5–20, 2002.

F. Peng, K. Tang, G. Chen, and X. Yao. Multi-start jade with knowledge transfer for numerical optimization. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 1889–1895. IEEE, 2009.

L. Peng and Y. Wang. Differential evolution using uniform-quasi-opposition for initializing the population. *Information Technology Journal*, 9(8):1629–1634, 2010.

L. Peng, Y. Wang, and G. Dai. Ude: differential evolution with uniform design. In *Parallel Architectures, Algorithms and Programming (PAAP), 2010 Third International Symposium on*, pages 239–246. IEEE, 2010.

L. Peng, Y. Wang, G. Dai, and Z. Cao. A novel differential evolution with uniform design for continuous global optimization. *Journal of Computers*, 7(1):3–10, 2012.

X. Peng and Y. Wu. Enhancing cooperative coevolution with selective multiple populations for large-scale global optimization. *Complexity*, 2018, 2018.

J. Pérez-Rodríguez, A. G. Arroyo-Peña, and N. García-Pedrajas. Simultaneous instance and feature selection and weighting using evolutionary computation: Proposal and study. *Applied Soft Computing*, 37:416–443, 2015.

F. Pezzella, G. Morganti, and G. Ciaschetti. A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research*, 35(10):3202–3212, 2008.

M. Pluhacek, R. Senkerik, I. Zelinka, and D. Davendra. Chaos pso algorithm driven alternately by two different chaotic maps–an initial study. In *IEEE Congress on Evolutionary Computation (CEC'13)*, pages 2444–2449. IEEE, 2013.

M. A. Potter and K. A. De Jong. A cooperative coevolutionary approach to function optimization. In *Proc. of International Conference on Parallel Problem Solving from Nature*, volume 2, pages 249–257, 1994.

M. J. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The computer journal*, 7(2):155–162, 1964.

M. J. Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. *Cambridge NA Report NA2009/06, University of Cambridge, Cambridge*, 2009.

K. Price, R. M. Storn, and J. A. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization*. Natural Computing Series. Springer Science & Business Media, 2006.

K. V. Price, R. M. Storn, and J. A. Lampinen. Differential evolution a practical approach to global optimization. 2005.

M. Ptashne. How Gene Activators Work. *Scientific American*, pages 40–47, January 1989.

A. K. Qin and X. Li. Differential evolution on the CEC-2013 single-objective continuous optimization testbed. In *IEEE Congress on Evolutionary Computation (CEC'13)*, pages 1099–1106. IEEE, 2013.

A. K. Qin and P. N. Suganthan. Self-adaptive differential evolution algorithm for numerical optimization. In *IEEE Congress on Evolutionary Computation (CEC'05)*, volume 2, pages 1785–1791. IEEE, 2005.

A. K. Qin, V. L. Huang, and P. N. Suganthan. Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE transactions on Evolutionary Computation*, 13(2):398–417, 2009.

N. V. Queipo, R. T. Haftka, W. Shyy, T. Goel, R. Vaidyanathan, and P. K. Tucker. Surrogate-based analysis and optimization. *Progress in aerospace sciences*, 41(1):1–28, 2005.

S. Rahnamayan and H. R. Tizhoosh. Image thresholding using micro opposition-based differential evolution (micro-ode). In *Evolutionary Computation, 2008. CEC 2008 .(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 1409–1416. IEEE, 2008.

S. Rahnamayan and G. G. Wang. Investigating in scalability of opposition-based differential evolution. *WSEAS Trans Comput*, 7:1792–1804, 2008a.

S. Rahnamayan and G. G. Wang. Solving large scale optimization problems by opposition-based differential evolution (ode). *WSEAS Transactions on Computers*, 7(10):1792–1804, 2008b.

S. Rahnamayan and G. G. Wang. Center-based sampling for population-based algorithms. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 933–938. IEEE, 2009.

S. Rahnamayan, H. R. Tizhoosh, and M. M. Salama. Opposition-based differential evolution for optimization of noisy problems. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 1865–1872. IEEE, 2006.

S. Rahnamayan, H. R. Tizhoosh, and M. Salama. A novel population initialization method for accelerating evolutionary algorithms. *Computers & Mathematics with Applications*, 53(10):1605–1614, 2007a.

S. Rahnamayan, H. R. Tizhoosh, and M. M. Salama. Opposition-based differential evolution (ode) with variable jumping rate. In *Foundations of Computational Intelligence, 2007. FOCI 2007. IEEE Symposium on*, pages 81–88. IEEE, 2007b.

S. Rahnamayan, H. R. Tizhoosh, and M. M. Salama. Quasi-oppositional differential evolution. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 2229–2236. IEEE, 2007c.

S. Rahnamayan, H. R. Tizhoosh, and M. Salama. Opposition versus randomness in soft computing techniques. *Applied Soft Computing*, 8(2):906–918, 2008.

D. Rainville, M. Sebag, C. Gagné, M. Schoenauer, D. Laurendeau, et al. Sustainable cooperative coevolution with a multi-armed bandit. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 1517–1524. ACM, 2013.

V. Rashtchi, A. Bayat, and H. Vahedi. Adaptive step length bacterial foraging algorithm. In *Intelligent Computing and Intelligent Systems, 2009. ICIS 2009. IEEE International Conference on*, volume 1, pages 322–326. IEEE, 2009.

R. G. Regis. An initialization strategy for high-dimensional surrogate-based expensive black-box optimization. In *Modeling and Optimization: Theory and Applications*, pages 51–85. Springer, 2013.

Y. Ren and Y. Wu. An efficient algorithm for high-dimensional function optimization. *Soft Computing*, 17(6):995–1004, 2013.

Z. Ren, B. Pang, M. Wang, Z. Feng, Y. Liang, A. Chen, and Y. Zhang. Surrogate model assisted cooperative coevolution for large scale optimization. *Applied Intelligence*, 49 (2):513–531, 2019.

M. Richards and D. Ventura. Choosing a starting configuration for particle swarm optimization. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, volume 3, pages 2309–2312. IEEE, 2004.

L. M. Rios and N. V. Sahinidis. Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56(3): 1247–1293, 2013.

K. Robbins, W. Zhang, J. Bertrand, and R. Rekaya. The ant colony algorithm for feature selection in high-dimension gene expression data for disease classification. *Mathematical medicine and biology: a journal of the IMA*, 24(4):413–426, 2007.

Y. Rojas and R. Landa. Towards the use of statistical information and differential evolution for large scale global optimization. In *Electrical Engineering Computing Science and Automatic Control (CCE), 2011 8th International Conference on*, pages 1–6. IEEE, 2011.

Y. Saka, M. Gunzburger, and J. Burkardt. Latinized, improved lhs, and cvt point sets in hypercubes. *International Journal of Numerical Analysis and Modeling*, 4(3-4):729–743, 2007.

H. Salehinejad, R. Zadeh, R. Liscano, and S. Rahnamayan. 3d localization in large-scale wireless sensor networks: A micro-differential evolution approach. In *Personal, Indoor, and Mobile Radio Communication (PIMRC), 2014 IEEE 25th Annual International Symposium on*, pages 1824–1828. IEEE, 2014.

R. Salomon. Reevaluating genetic algorithm performance under coordinate rotation of benchmark functions - a survey of some theoretical and practical aspects of genetic algorithms. *BioSystems*, 39:263–278, 1995.

A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, and S. Tarantola. *Global sensitivity analysis: the primer*. John Wiley & Sons, 2008.

G. Sanchita and D. Anindita. Evolutionary algorithm based techniques to handle big data. In *Techniques and Environments for Big Data Analysis*, pages 113–158. Springer, 2016.

C. P. Sankar, S. Asharaf, and K. S. Kumar. Learning from bees: An approach for influence maximization on viral campaigns. *PloS one*, 11(12):e0168125, 2016.

M. L. Sanyang and A. Kaban. Multivariate cauchy EDA optimisation. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 449–456. Springer, 2014.

M. L. Sanyang and A. Kabán. Heavy tails with parameter adaptation in random projection based continuous eda. In *Evolutionary Computation (CEC), 2015 IEEE Congress on*, pages 2074–2081. IEEE, 2015.

P. Sarma, W. H. Chen, et al. Efficient well placement optimization with gradient-based algorithms and adjoint models. In *Intelligent Energy Conference and Exhibition*. Society of Petroleum Engineers, 2008.

M. Sato and Y. Fukuyama. Total optimization of smart community using sequence-based deterministic initialization and k-means based initial searching points generation. In *2017 19th International Conference on Intelligent System Application to Power Systems (ISAP)*, pages 1–6. IEEE, 2017.

A. Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

R. Senkerik, D. Davendra, I. Zelinka, and Z. Oplatkova. Influence of chaotic dynamics on the performance of evolutionary algorithms-an initial study. In *AIP Conference Proceedings*, volume 1479, page 627, 2012.

R. Senkerik, M. Pluhacek, Z. K. Oplatkova, D. Davendra, and I. Zelinka. Investigation on the differential evolution driven by selected six chaotic systems in the task of reactor geometry optimization. In *IEEE Congress on Evolutionary Computation (CEC'13)*, pages 3087–3094. IEEE, 2013.

S. Shan and G. G. Wang. Metamodeling for high dimensional simulation-based design problems. *Journal of Mechanical Design*, 132(5):051009, 2010a.

S. Shan and G. G. Wang. Survey of modeling and optimization strategies to solve high-dimensional design problems with computationally-expensive black-box functions. *Structural and Multidisciplinary Optimization*, 41(2):219–241, 2010b.

Y.-W. Shang and Y.-H. Qiu. A note on the extended rosenbrock function. *Evolutionary Computation*, 14(1):119–126, March 2006.

M. Sharma and S. Tyagi. Novel knowledge based selective tabu initialization in genetic algorithm. *International Journal*, 3(5), 2013.

D. J. Sheskin. *Handbook of parametric and nonparametric statistical procedures*. CRC Press, 2003.

L. Shi and K. Rasheed. A survey of fitness approximation methods applied in evolutionary algorithms. *Computational intelligence in expensive optimization problems*, pages 3–28, 2010.

H. Shimamoto, N. Murayama, A. Fujiwara, and J. Zhang. Evaluation of an existing bus network using a transit network optimisation model: a case study of the hiroshima city bus network. *Transportation*, 37(5):801–823, 2010.

H. K. Singh and T. Ray. Divide and conquer in coevolution: A difficult balancing act. *Agent-Based Evolutionary Search*, pages 117–138, 2010.

I. H. Sloan and S. Joe. *Lattice methods for multiple integration*. Oxford University Press, 1994.

J. Smith and T. C. Fogarty. An adaptive poly-parental recombination strategy. In *Selected Papers from AISB Workshop on Evolutionary Computing*, pages 48–61, London, UK, 1995. Springer-Verlag. linkage evolving genetic operator (LEGO).

L. Smith. *Chaos: a very short introduction*. Oxford University Press, 2007.

J. Snyman. *Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms*, volume 97. Springer Science & Business Media, 2005.

J. Sobieszczanski-Sobieski and R. T. Haftka. Multidisciplinary aerospace design optimization: Survey of recent developments. *Structural Optimization*, 14(1):1–23, August 1997. ISSN 1615-1488. doi: 10.1007/BF01197554. URL http://dx.doi.org/10.1007/BF01197554.

I. M. Sobol. On quasi-monte carlo integrations. *Mathematics and Computers in Simulation*, 47(2):103–112, 1998.

I. M. Sobol. Theorems and examples on high dimensional model representation. *Reliability Engineering & System Safety*, 79(2):187–193, 2003.

F. J. Solis and R. J.-B. Wets. Minimization by random search techniques. *Mathematics of operations research*, 6(1):19–30, 1981.

J. Soto. Statistical testing of random number generators. In *Proceedings of the 22nd National Information Systems Security Conference*, volume 10, page 12. NIST Gaithersburg, MD, 1999.

T. Sousa, T. Soares, H. Morais, R. Castro, and Z. Vale. Simulated annealing to handle energy and ancillary services joint management considering electric vehicles. *Electric Power Systems Research*, 136:383–397, 2016.

S. Srinivasan and S. Ramakrishnan. Evolutionary multi objective optimization for rule mining: a review. *Artificial Intelligence Review*, 36(3):205–248, 2011.

R. Storn and K. Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

C. Sun, Y. Jin, J. Zeng, and Y. Yu. A two-layer surrogate-assisted particle swarm optimization algorithm. *Soft computing*, 19(6):1461–1475, 2015a.

C. Sun, J. Ding, J. Zeng, and Y. Jin. A fitness approximation assisted competitive swarm optimizer for large scale expensive optimization problems. *Memetic Computing*, pages 1–12, 2016.

Y. Sun, M. Kirley, and S. K. Halgamuge. Extended differential grouping for large scale global optimization with direct and indirect variable interactions. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 313–320. ACM, 2015b.

Y. Sun, X. Li, A. Ernst, and M. N. Omidvar. Decomposition for large-scale optimization problems with overlapping components. In *2019 IEEE Congress on Evolutionary Computation (CEC)*, pages 326–333. IEEE, 2019.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. Cambridge Univ Press, 1998.

T. Takahama and S. Sakai. Large scale optimization by differential evolution with landscape modality detection and a diversity archive. In *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pages 1–8. IEEE, 2012.

K. Tang, X. Yao, P. N. Suganthan, C. MacNish, Y. P. Chen, C. M. Chen, , and Z. Yang. Benchmark functions for the CEC'2008 special session and competition on large scale global optimization. Technical report, Nature Inspired Computation and Applications Laboratory, USTC, China, 2007. URL http://nical.ustc.edu.cn/cec08ss.php.

K. Tang, X. Li, P. N. Suganthan, Z. Yang, and T. Weise. Benchmark functions for the CEC'2010 special session and competition on large-scale global optimization. Technical report, Nature Inspired Computation and Applications Laboratory, USTC, China, 2009.

M. G. C. Tapia and C. A. Coello Coello. Applications of multi-objective evolutionary algorithms in economics and finance: A survey. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 532–539. IEEE, 2007.

M. Tezuka, M. Munetomo, and K. Akama. Linkage identification by nonlinearity check for real-coded genetic algorithms. In *Proc. of Genetic and Evolutionary Computation Conference*, volume 3103 of *Lecture Notes in Computer Science*, pages 222–233. Springer, 2004.

S. A. Thomas and Y. Jin. Reconstructing biological gene regulatory networks: where optimization meets big data. *Evolutionary Intelligence*, 7(1):29–47, 2014.

T. Tometzki and S. Engell. Systematic initialization techniques for hybrid evolutionary algorithms for solving two-stage stochastic mixed-integer programs. *Evolutionary Computation, IEEE Transactions on*, 15(2):196–214, 2011.

A. O. Topal, O. Altun, and Y. E. Yildiz. Micro bat algorithm for high dimensional optimization problems. *International Journal of Computer Applications*, 122(12), 2015.

G. A. Trunfio. Adaptation in cooperative coevolutionary optimization. In *Adaptation and Hybridization in Computational Intelligence*, volume 18, pages 91–109. Springer, 2015.

L.-Y. Tseng and C. Chen. Multiple trajectory search for large scale global optimization. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 3052–3059. IEEE, 2008. MTSLS.

N. Q. Uy, N. X. Hoai, R. McKay, and P. M. Tuan. Initialising pso with randomised low-discrepancy sequences: the comparative results. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 1985–1992. IEEE, 2007.

F. van den Bergh and A. P. Engelbrecht. A cooperative approach to particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):225–239, 2004.

K. Veeramachaneni and L. A. Osadciw. Dynamic sensor management using multi objective particle swarm optimizer. Technical report, Department of Electrical Engineering and Computer Science, Syracuse University, 2004.

J. Vermorel and M. Mohri. Multi-armed bandit algorithms and empirical evaluation. In *Machine learning: ECML 2005*, pages 437–448. Springer, 2005.

R. Vershynin. Introduction to the non-asymptotic analysis of random matrices. *arXiv preprint arXiv:1011.3027*, 2010.

A. F. Villaverde, J. A. Egea, and J. R. Banga. A cooperative strategy for parameter estimation in large scale systems biology models. *BMC systems biology*, 6(1):75, 2012.

E. I. Vlahogianni. Computational intelligence and optimization for transportation big data: challenges and opportunities. In *Engineering and Applied Sciences Optimization*, pages 107–128. Springer, 2015.

J. Walker. Hotbits: genuine random numbers. *HotBits: Genuine Random Numbers. September*, 2006.

C. Wang, W. Chen, and Y. Wang. Scalable influence maximization for independent cascade model in large-scale social networks. *Data Mining and Knowledge Discovery*, 25(3):545, 2012a.

H. Wang, Z. Wu, J. Wang, X. Dong, S. Yu, and C. Chen. A new population initialization method based on space transformation search. In *Natural Computation, 2009. ICNC'09. Fifth International Conference on*, volume 5, pages 332–336. IEEE, 2009.

H. Wang, Z. Wu, S. Rahnamayan, Y. Liu, and M. Ventresca. Enhancing particle swarm optimization using generalized opposition-based learning. *Information Sciences*, 181 (20):4699–4714, 2011.

H. Wang, S. Rahnamayan, and Z. Wu. Parallel differential evolution with self-adapting control parameters and generalized opposition-based learning for solving high-dimensional optimization problems. *Journal of Parallel and Distributed Computing*, 73 (1):62–73, 2013.

H. Wang, Y. Jin, and J. Doherty. Committee-based active learning for surrogate-assisted particle swarm optimization of expensive problems. *IEEE Transactions on Cybernetics*, 2017.

X. Wang and I. H. Sloan. Low discrepancy sequences in high dimensions: How well are their projections distributed? *Journal of Computational and Applied Mathematics*, 213 (2):366–386, 2008.

Y. Wang and B. Li. A restart univariate estimation of distribution algorithm: sampling under mixed gaussian and lévy probability distribution. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 3917–3924. IEEE, 2008.

Y. Wang, Z. Cai, and Q. Zhang. Enhancing the search ability of differential evolution through orthogonal crossover. *Information Sciences*, 185(1):153–177, 2012b.

Z. Wang, W. Huang, and L. Yan. Multidisciplinary design optimization approach and its application to aerospace engineering. *Chinese science bulletin*, 59(36):5338–5353, 2014.

M. Weber, F. Neri, and V. Tirronen. Shuffle or update parallel differential evolution for large-scale optimization. *Soft Computing*, 15(11):2089–2107, 2011.

K. Weicker and N. Weicker. On the improvement of coevolutionary optimizers by learning variable interdependencies. In *Proc. of IEEE Congress on Evolutionary Computation*, pages 1627–1632. IEEE Press, 1999.

T. Weise, R. Chiong, and K. Tang. Evolutionary Optimization: Pitfalls and Booby Traps. *Journal of Computer Science and Technology (JCST)*, 27(5):907–936, 2012. Special Issue on Evolutionary Computation.

H. Weyl. Über die gleichverteilung von zahlen mod. eins. *Mathematische Annalen*, 77(3): 313–352, 1916.

T. Wimalajeewa and S. K. Jayaweera. Optimal power scheduling for correlated data fusion in wireless sensor networks via constrained pso. *IEEE Transactions on Wireless Communications*, 7(9), 2008.

F. Xhafa and A. Abraham. *Metaheuristics for scheduling in distributed computing environments*, volume 146. Springer, 2008.

Q. Xu, N. Wang, and R. Fei. Influence of dimensionality and population size on opposition-based differential evolution using the current optimum. *Information Technology Journal*, 12:105–112, 2013.

Q. Xu, L. Wang, N. Wang, X. Hei, and L. Zhao. A review of opposition-based learning from 2005 to 2012. *Engineering Applications of Artificial Intelligence*, 2014.

B. Xue, M. Zhang, W. N. Browne, and X. Yao. A survey on evolutionary computation approaches to feature selection. *IEEE Transactions on Evolutionary Computation*, 20 (4):606–626, 2016.

M. Yang, M. N. Omidvar, C. Li, X. Li, Z. Cai, B. Kazimipour, and X. Yao. Efficient resource allocation in cooperative co-evolution for large-scale global optimization. *IEEE Transactions on Evolutionary Computation*, PP(1):1–1, 2017.

M. Yang, A. Zhou, C. Li, J. Guan, and X. Yan. CCFR2: A more efficient cooperative co-evolutionary framework for large-scale global optimization. *Information Sciences*, 2019.

Z. Yang, K. Tang, and X. Yao. Differential evolution for high-dimensional function optimization. In *IEEE Congress on Evolutionary Computation (CEC'07)*, pages 3523–3530. IEEE, 2007a.

Z. Yang, X. Yao, and J. He. Making a difference to differential evolution. In *Advances in metaheuristics for hard optimization*, pages 397–414. Springer, 2007b.

Z. Yang, K. Tang, and X. Yao. Large scale evolutionary optimization using cooperative coevolution. *Information Sciences*, 178(15):2986–2999, August 2008a. DECC-G.

Z. Yang, K. Tang, and X. Yao. Multilevel cooperative coevolution for large scale optimization. In *Proc. of IEEE Congress on Evolutionary Computation*, pages 1663–1670. IEEE, June 2008b. MLCC.

Z. Yang, K. Tang, and X. Yao. Self-adaptive differential evolution with neighborhood search. In *IEEE Congress on Evolutionary Computation (CEC'08)*, pages 1110–1116. IEEE, 2008c. SaNSDE.

Z. Yang, K. Tang, and X. Yao. Scalability of generalized adaptive differential evolution for large-scale continuous optimization. *Soft Computing*, 15(11):2141–2155, 2011. GADE.

C. Yanguang, M. Zhang, and C. Hao. A hybrid chaotic quantum evolutionary algorithm. In *Intelligent Computing and Intelligent Systems (ICIS), 2010 IEEE International Conference on*, volume 2, pages 771–776. IEEE, 2010.

D. Yazdani, M. N. Omidvar, J. Branke, T. T. Nguyen, and X. Yao. Scaling up dynamic optimization problems: A divide-and-conquer approach. *IEEE Transactions on Evolutionary Computation*, 2019.

H. Yi, Q. Duan, and T. W. Liao. Three improved hybrid metaheuristic algorithms for engineering design optimization. *Applied Soft Computing*, 13(5):2433–2444, 2013.

Y. E. Yildiz, O. Altun, and A. O. Topal. Computational chemotaxis in micro bacterial foraging optimization for high dimensional problems: a comparative study on numerical benchmark. *International Journal of Computer Applications*, 124(4), 2015.

D. Zaharie. Critical values for the control parameters of differential evolution algorithms. In *Proceedings of MENDEL*, pages 62–67, 2002.

A. Zamuda, J. Brest, B. Boskovic, and V. Zumer. Large scale global optimization using differential evolution with self-adaptation and cooperative co-evolution. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 3718–3725, 2008.

A. Zamuda, J. Brest, B. Bošković, and V. Žumer. Differential evolution for parameterized procedural woody plant models reconstruction. *Applied Soft Computing*, 11(8):4904–4912, 2011.

S. Zaremba. Good lattice points, discrepancy, and numerical integration. *Annali di matematica pura ed applicata*, 73(1):293–317, 1966.

I. Zelinka, R. Senkerik, and M. Pluhacek. Do evolutionary algorithms indeed require randomness? In *IEEE Congress on Evolutionary Computation (CEC'13)*, pages 2283–2289. IEEE, 2013.

G. Zhang, L. Gao, and Y. Shi. An effective genetic algorithm for the flexible job-shop scheduling problem. *Expert Systems with Applications*, 38(4):3563–3573, 2011.

J. Zhang and A. C. Sanderson. JADE: adaptive differential evolution with optional external archive. *IEEE Transactions on Evolutionary Computation*, 13(5):945–958, 2009.

M. Zhang, W. Zhang, and Y. Sun. Chaotic co-evolutionary algorithm based on differential evolution and particle swarm optimization. In *Automation and Logistics, 2009. ICAL'09. IEEE International Conference on*, pages 885–889. IEEE, 2009.

S. Zhao, J. Liang, P. Suganthan, and M. Tasgetiren. Dynamic multi-swarm particle swarm optimizer with local search for large scale global optimization. In *Proc. of IEEE Congress on Evolutionary Computation*, pages 3845 –3852, June 2008. Large-Scale DMS-PSO.

S.-Z. Zhao, P. N. Suganthan, and S. Das. Self-adaptive differential evolution with multi-trajectory search for large-scale optimization. *Soft Computing*, 15(11):2175–2185, 2011.

H. Zheng, Y. Zheng, and P. Li. Sine-map chaotic pso-based neural network predictive control for deployable space truss structures. In *Industrial Electronics (ISIE), 2013 IEEE International Symposium on*, pages 1–5, May 2013. doi: 10.1109/ISIE.2013.6563726.

Z. Zhou, Y. S. Ong, P. B. Nair, A. J. Keane, and K. Y. Lum. Combining global and local surrogate models to accelerate evolutionary optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 37(1):66–76, 2007.

E. Zitzler. *Evolutionary algorithms for multiobjective optimization: Methods and applications*, volume 63. Shaker Ithaca, 1999.